An Anatomy of Graph Neural Networks Going Deep via the Lens of Mutual Information: Exponential Decay vs. Full Preservation

Nezihe Merve Gürel[1], **Hansheng Ren**[2, 3], **Yujing Wang**[2], **Hui Xue**[2], **Yaming Yang**[2], **Ce Zhang**[1]
[1]ETH Zurich, [2]Microsoft Research Asia, [3]University of Chinese Academy of Sciences

Abstract

Graph Convolutional Network (GCN) has attracted intensive interests recently, with a major limitation that it often cannot benefit from using a deep architecture, while traditional CNN and an alternative Graph Neural Network architecture, namely GraphCNN, generally achieve better quality with a deeper neural architecture. *How can we explain this phenomenon?* In this paper, we take the first step towards answering this question. We first conduct a systematic empirical study on the accuracy of GCN, GraphCNN, and ResNet-18 on 2D images and identified relative importance of different factors in the architectural design. This inspired a novel theoretical analysis on the mutual information between the input and the output after l GCN/ GraphCNN layers. We identified regimes in which GCN suffers exponentially fast "information lose", and GraphCNN has a better capability of preserving sufficient information at the output layer.

Introduction

Extending convolutional neural networks (CNN) over images to graphs has attracted intense interest recently. One early attempt is the GCN model proposed by Kipf and Welling (2016a). When applying GCN to many practical applications, however, one discrepancy lingers — although traditional CNN usually gets higher accuracy when it goes deeper, GCN, as a natural extension of CNN, does not seem to benefit much from going deeper by stacking multiple layers together.

This phenomenon has been the focus of multiple recent papers (Li, Han, and Wu, 2018; Li et al., 2019; Oono and Suzuki, 2019). On the theoretical side, Li, Han, and Wu (2018) and Oono and Suzuki (2019) identified the problem as *oversmoothing* — under certain conditions, when multiple GCN layers are stacked together, the output will converge to a region that is independent of weights and inputs. On the empirical side, Li et al. (2019) showed that many techniques that were designed to train a deep CNN, e.g., the skip connections in ResNet (He et al., 2016a), can make it easier for GCN to go deeper. Yet two questions remain: *Does there exist a set of techniques that make GCN at least as powerful (in terms of accuracy) as state-of-the-art CNN?* If so, *can we prove that the oversmoothing problem of GCN will cease to exist after these techniques are implemented?*

In this paper, we conduct a systematic empirical study and a novel theoretical analysis as the first step to answering these questions, by putting GCN and CNNs on the same ground.

Pillar 1. Empirical Study Our work builds upon GraphCNN Such et al. (2017). Let $\mathbf{A}$ be adjacency matrix of a graph, and $\mathbf{W}$ be learned weight matrix. For an input $\mathbf{X}$, the GCN layer response has the form $\mathbf{AXW}$, whereas in GraphCNN, the adjacency matrix $\mathbf{A}$ is decomposed into additive matrices: $\mathbf{A} = \sum_i \mathbf{A}_i$ and the layer response is of the form $\sum_i \mathbf{A}_i\mathbf{XW}_i$.

Under one decomposition strategy, a GraphCNN layer recovers a CNN layer (We refer to Section A in the supplementary material for more details). Although it is not surprising that GraphCNN can match the accuracy of CNN under a certain decomposition strategy, we ask: *How fundamental is this decomposition step? Can GCN match the accuracy of ResNet empirically if we integrate all standard techniques and tricks, such as stride, skip connection, and average pooling?*

In our empirical study, we convert CIFAR-10 images into an equivalent graph representation, and compare GCN, GraphCNN, and ResNet with the same depths. For each model, we study the impact of (1) stride, (2) skip connection, and (3) pooling.

Although stride, skipped connection and pooling significantly improve the accuracy of GCN as formerly noted by Li et al. (2019), we observe that *the decomposition step in GraphCNN is fundamental and sufficient* for GCN to achieve state-of-the-art accuracy of ResNet.

Pillar 2. Theoretical Analysis Motivated by this empirical result, we then focus on understanding the theoretical property of the decomposition step in GraphCNN. Specifically, we ask: *Can we precisely analyze the benefits introduced by graph decomposition in GraphCNN, compared with GCN?*

This question poses three challenges that existing analyses on GCN oversmoothing (Li, Han, and Wu, 2018; Oono and Suzuki, 2019) cannot handle: (1) While both frameworks reason about how closely the GCN output after l layers will approach a region that is independent of weights and inputs, to answer our question we need to reason not only

geometrically but also more direct notion of *utility* — being close to a bad region *geometrically* is definitely *bad*, but being far away from it does not necessarily mean it is *better* (See Section 4). (2) While both theoretical frameworks provide an *upper bound* of the distance, however, the upper bound itself is not enough to answer our question. (3) None of the existing analysis on GCN considered the impact of graph decomposition in GraphCNN.

In this paper, we conduct a theoretical analysis that directly reasons about the *mutual information* between the output after l layers and the input. We show that under certain conditions, (1) the MI (Mutual Information) after l GCN layers with (parametric) ReLUs (1.a) converges to 0 exponentially fast, (1.b) perfectly preserves all information in the input, (2) the MI after l GraphCNN layers with (parametric) ReLUs perfectly preserves all information in the input. *More importantly, compared with GCN, GraphCNN perfectly preserves information at its output in a much larger regime of weights, largely because of the decomposition structure introduced in GraphCNN.*

Putting these results together, we provide a precise theoretical description of the power of graph decomposition introduced in GraphCNN. To the best of our knowledge, this is one of the first results of its form.

Moving Forward. Our analysis brings up a natural question: *"How can we choose the decomposition strategy in GraphCNN? Moreover, can we learn it automatically?"* We believe that this offers an interesting direction for further work and hope that this paper can help to facilitate future endeavours in this direction.

Related Work

Deep neural networks on graphs has attracted intense interest in recent years. Motivated by the success of (Krizhevsky, Sutskever, and Hinton, 2012), (Bruna et al., 2013) models the filters as learnable parameters based on the spectrum of the graph Laplacian. ChebNet (Defferrard, Bresson, and Vandergheynst, 2016) reduces computation complexity by approximating the filter with Chebyshev polynomials of the diagonal matrix of eigenvalues; Graph Convolutional Network (GCN) (Kipf and Welling, 2016b) goes further, introducing a first-order approximation of ChebNet and making several simplifications. GCN and its variants have been widely applied in various graph-related applications, including semantic relationship recognition (Xu et al., 2017), graph-to-sequence learning (Beck, Haffari, and Cohn, 2018), traffic forecasting (Li et al., 2017) and molecule classification (Such et al., 2017). Although GCN and its variants have achieved promising results on various graph applications, it cannot obtain better performance with the increase of network depths. For instance, Kipf and Welling (2016a) show that a two-layer GCN would achieve peak performance, and stacking more layers cannot bring any improvement. Rahimi, Cohn, and Baldwin (2018) develop a highway GCN for user geolocation in social media graphs, in which highway gates were added between layers to facilitate gradient flow. Even with these gates, the authors demonstrate performance degradation after six layers of depth. This phenomena is counter-intuitive and blocks GCN-style models from making further improvements. There are plenty of results (Zhou et al., 2018; Wu et al., 2019b) trying to figure out the reasons and provide workarounds. Wu et al. (2019a) hypothesize that nonlinearity between GCN layers is not critical, which essentially implies that the deep GCN model lacks sufficient expressive ability because it is a linear model. In addition, Li et al. (2019) show that the techniques such as skip connection in ResNet can help GCN to train deeper; however, they do not provide an empirical study of whether this modification is enough for GCN to match the quality of state-of-the-art CNNs (e.g., ResNet) on images.

Li, Han, and Wu (2018) show that GCN is a special form of Laplacian smoothing, and under certain conditions, the features of vertices within each connected component of the graph will converge to the same values by repeatedly applying Laplacian smoothing. Therefore, the oversmoothing property of GCN will make the features indistinguishable and thus hurt the classification accuracy. Oono and Suzuki (2019) also conduct more engaged theoretical analysis. In this paper, however, we directly reason about mutual information and we are more interested in understanding the decomposition structure in GraphCNN instead of the oversmoothing property of GCN. Our work also builds on Such et al. (2017), which proposes GraphCNN that consists of multiple adjacency matrices. As shown by Such et al. (2017), this formulation is more expressive than CNN. Here we use the same framework but focus on providing a novel empirical study and theoretical analysis to understand the behavior of GCN and the power of graph decomposition in GraphCNN.

Preliminaries

Hereafter, scalars will be written in italics, and matrices in bold upper-case letters.

Let $G = (V, E)$ be an undirected graph with a vertex set $v_i \in V$ and set of edges $e_{i,j} \in E$. We refer to individual elements of v_i as nodes, and $\mathbf{x}_i \in \mathbb{R}^d$ associated with each v_i as features. We denote the node feature attributes by $\mathbf{X} \in \mathbb{R}^{n \times d}$ whose rows are given by $\mathbf{x}_i$. The adjacency matrix $\mathbf{A}$ (weighted or binary) is derived as an $n \times n$ matrix with $(\mathbf{A})_{i,j} = e_{i,j}$ if $e_{i,j} \in E$, and $(\mathbf{A})_{i,j} = 0$ elsewhere.

We define the following operator $f : \mathbb{R}^n \to \mathbb{R}^n$ that is composed of (1) a linear function parameterized by the adjacency matrix $\mathbf{A}$ and a weight matrix at layer $i+1$ $\mathbf{W}^{(i+1)}$, and (2) an activation function as parametric ReLU such that $\sigma : x \to \max(x, ax)$ with $a \in (0, 1)$ that applies following the linear transformation of previous layer element-wise. Given the input matrix $\mathbf{X}$, let $\mathbf{Y}^{(0)} = \mathbf{X}$. Each layer of the network maps it to an output vector of the same shape: $\mathbf{Y}^{(i+1)} = f_{\mathbf{A},\mathbf{W}^{(i+1)}}(\mathbf{Y}^{(i)}) = \sigma(\mathbf{A}\mathbf{Y}^{(i)}\mathbf{W}^{(i+1)})$.

Let now $\mathbf{A} \in \mathbb{R}^{n \times n}$ be decomposed into K additive $n \times n$ matrices such that $\mathbf{A} = \sum_{k=1}^{K} \mathbf{A}_k$. The layer-wise propagation rule becomes $\mathbf{Y}^{(i+1)} = g_{\mathbf{A}_k,\mathbf{W}_k^{(i+1)}}(\mathbf{X}) = \sigma\big(\sum_{k=1}^{K} \mathbf{A}_k\mathbf{X}\mathbf{W}_k^{(i+1)}\big)$.

Empirical Study

In this section, we conduct a systematic empirical study to understand the impact of different types of layers and different techniques/tricks. We observe that (1) the techniques designed for CNN can also improve the accuracy of GCN significantly, which is consistent with previous work (Li et al., 2019); however, (2) the graph decomposition step introduced in GraphCNN is a fundamental step whose impact cannot be offset, even if we apply *all* techniques. This motivates our theoretical study in the next section which tries to theoretically describe the impact of graph decomposition.

Experimental Setup

We constructed an equivalent graphical representation of the CIFAR-10 images treating each pixel as a node in the graph, and the surrounding pixels in 9 directions (including itself) as neighboring nodes in order to mimic the behavior of a 3×3 convolution. Consisting of 60000 images of 32×32 pixels with RGB channels: each CIFAR-10 image corresponds to a graph with 1024 (32×32) vertices, each of which connects to the 8 neighbors plus a self-connection.

Noting that (1) a deep CNN achieves state-of-the-art quality for image classification, whereas a deep GCN cannot benefit from deep architectures; (2) a deep GraphCNN, with all optimization tricks and the right graph decomposition strategy, also matches the accuracy of state-of-the-art CNNs as it has an equivalent expressive power as that of CNN, *The goal of our study is then to understand the relative impact of graph decomposition and useful tricks designed for CNNs.*

Model Architectures. We compare three model architectures: CNN, GCN, and GraphCNN:

1. CNN (Krizhevsky, Sutskever, and Hinton, 2012) The architecture is stacked by 3×3 convolution layers. The input channel of the first CNN layer is 3 (including RGB) and the output channel is set as 128. All the input and output channels of the succeeding convolution layers are 128.

2. GCN (Kipf and Welling, 2016b) We treat all edges in the graph equally and leverage a similar network architecture as CNN. The only difference is that we replace each 3×3 convolution layer with a GCN layer.

3. GraphCNN (Such et al., 2017) We replace each convolution layer with a GraphCNN layer. Specifically, we decompose the adjancency matrix $\mathbf{A}$ into 9 submatrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_9$. For two arbitrary pixels (i, j) and (m, n), we set the edges $e(i, j, m, n) = 1$ of each submatrix $\mathbf{A}_i$ when the following equation holds; otherwise the corresponding edges are set as zero in that matrix. An illustration and further details are provided in Figure 4 and Section A in the supplementary material.

For each of these three architectures, we focus on the following techniques:

1. Original. Applying (graph) convolution operations in each layer with $stride = 1$ and with no skip of connections, we reshape the 2D image to a 1D embedding vector and add a fully connected layer with Softmax activation on the top to generate classification results at the last layer, where all hidden size is set to 128.

2. Stride. The stride of each layer is aligned with ResNet-18. Specifically for the 9th, 13th, and 17th layer, we apply $stride = 2$, and both the length and width of the original image will be halved. We follow a common strategy with the hidden size is doubled upon a stride operation (original hidden size = 128). To imitate the stride behavior for GCN and GraphCNN, we perform convolution first, then choose the nodes that will be reserved by the strides to construct a new grid graph corresponding to the smaller image.

3. Stride+Skip. We add skip connections between the corresponding layers following the standard architecture of ResNet-18 (See XXX for more details). Other configurations are kept the same with the **Stride** setting.

4. Stride+Skip+AP. The network architecture of the 17-layer CNN looks similar to ResNet-18 except that it does not adopt average pooling before the final fully connected layer. To align with ResNet-18, we also compare the models in the architectures with average pooling on the top such that the 17-layer CNN exactly matches the network architecture of a standard ResNet-18.

We used the standard data argument method in all experiments, including random cropping and random flipping (Simonyan and Zisserman, 2014). All experiments were trained using SGD with Momentum (Ruder, 2016), where the momentum ratio was set as 0.9 and weight decay factor as 1×10^{-5}. We chose the best learning rate via grid search and initialized the network parameters using Xavier (Glorot and Bengio, 2010). Similar to a standard ResNet (He et al., 2016b), we did not use dropout in the experiments. All experiments were conducted on a Tesla P100 with 16GB GPU memory.

Results and Discussions

We observe from the results demonstrated in Figure 1 and Figure 2 that GraphCNN is as powerful as CNN, even without pooling layers. For architecture depth being 1, CNN and GraphCNN significantly outperform GCN. As the depth increases, GCN still underperforms GraphCNN. However, without any tricks designed for CNN (Figure 1.a&2.a), when the depth becomes greater than 9 layers, GCN gets no better, while the CNN and GraphCNN counterparts still benefit from deeper architectures. GCN can seemingly go deep to some extent, but the optimal depth and accuracy being smaller.Stride is then set to 2 for some layers following ResNet (Figure 1.b&2.b), and the performance of all models has been improved. Particularly for GCN, the test performance has been improved from 57.1% to 60.2%. Moreover, models exhibit similar relative trends as before. We further add skip connections (Figure 1.c&2.c) and observe that the residual connections do have a positive effect for training a deep GCN network, improving the test score from 60.2% to 64.4%. However, it is still well behind the state-of-the-art results from CNN and GraphCNN. Finally, we add an average pooling layer at the end to fully match the architecture of a state-of-the-art ResNet (Figure 1.d&2.d). The average pooling layer provides improvement on GCN. Yet, a significant gap between GCN and GraphCNNs/CNNs exists — the GCN model suffers from severe overfitting, obtaining only 72.8% accuracy with a 17-layer architecture, even though the training accuracy achieved 94.0%.

We conclude that *the graph decomposition introduced in*

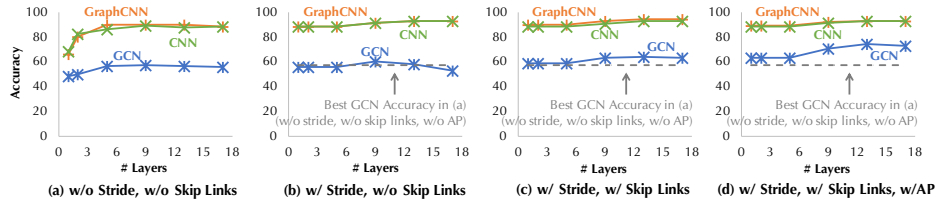


Figure 1: Testing Accuracy on CIFAR-10.

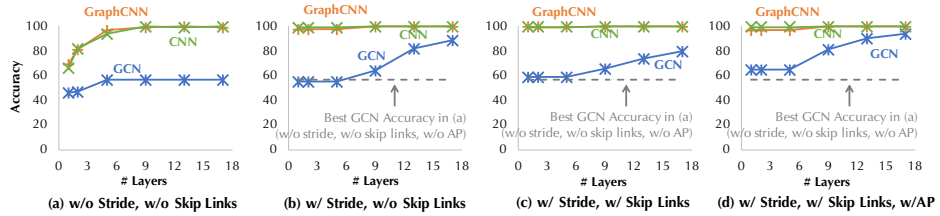


Figure 2: Training Accuracy on CIFAR-10.

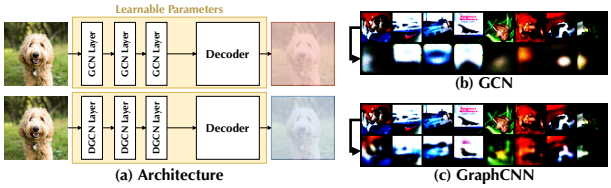


Figure 3: (a) The neural network architecture that illustrates the mutual information decay after three GCN layers or three GraphCNN layers. Intuitively, the decoder estimates the MI in a similar way as MINE. (b/c) Reconstructions of test images from the output after 3 GCN/GraphCNN layers. The first row is the input images and the second row is the output images of the decoder.

GraphCNN is fundamental: despite the desirable tricks often used for a deep ResNet architecture. GCN still has an almost 20 percent point gap compared with CNN, whereas a decomposition strategy lifts the accuracy to that of CNN.

Empirical Information Loss

To compare mutual information between the input and output layers of GCN as well as GraphCNN, we adapt a methodology similar to Belghazi et al. (2018) and the architecture illustrated in Figure 7(a) as the *proxy* of the MI after l layers. In order to measure the MI after l layers, we take the first l GCN/GraphCNN layers and add a fully connected layer that shrinks the hidden unit size, followed by a decoder, a single fully connected layer, that reconstructs the hidden unit size to the input. We measure the reconstruction error modeled by l_1 loss (Janocha and Czarnecki, 2017), and train the network end-to-end and optimize the hyperparameters by Random Search (Bergstra and Bengio, 2012).

Figure 7(b, c) illustrates the reconstruction results after $l = 3$ layers. The overall reconstruction error of GraphCNN (0.781) outperforms that of GCN (0.818) significantly. The GCN reconstruction results clearly show the over-smoothing phenomenon, previously introduced by (Li, Han, and Wu, 2018; Oono and Suzuki, 2019). GraphCNN is, on the other hand, able to preserve significantly more information (if the training objective is to maintain as much information as possible).

Decomposition Strategy Matters

Although fully answering the question *How does a different decomposition strategy impact the accuracy?* in a single paper would be a forlorn hope, we note the fascinating result from a simple experiment. We tested GraphCNN with three random decomposition strategies, and observed that the accuracy with a 17-layer GraphCNN drops significantly. Specifically, in the **Stride+Skip+AP** setting, the accuracy drops from 93.2% to 83.8% (the average performance of three randomly decomposed GraphCNN), which indicates that the decomposition strategy does have a significant impact to the final performance. However, interestingly, the randomly decomposed GCN still outperforms the vanilla GCN layers (72.8% accuracy).

Moving Forward. We believe that the analysis on GraphCNN with a random graph decomposition, and its connection to random features are immediate future directions. Another interesting future direction would be to design a system abstraction to allow users to specify graph decompositions or even use some automatic approaches similar to NAS (Zoph and Le, 2016) to automatically search for the optimal graph decomposition strategies for GraphCNN.

Theoretical Analysis

The dramatic difference between GCN and GraphCNN can look quite counter-intuitive at the first glance. *Why can a simple decomposition of the adjacency matrix A have such a significant impact on both the accuracy and the preservation property of mutual information?* In this section, we provide a theoretical analysis of the mutual information between the lth layer of either network and the input.

Our theoretical analysis suggests that GraphCNN has a better data processing capability than that of GCN under the same characteristics of layer-wise weight matrices, justifying the observation that GraphCNN overcomes the *overcompression* introduced by GCN as we pile up more layers.

GCN

Here we investigate the regimes where GCN (1) does not benefit from going deeper, or (2) is guaranteed to preserve all information at its output, by analyzing the behaviour of mutual information between input and output layer of the

network at different depths. We relegate all the proofs to the Section B in the supplementary material.

Throughout the paper, we denote the vectorized input $\mathbf{X}$ and lth layer output $\mathbf{Y}^{(l)}$ by $\mathbf{x}$ and $\mathbf{y}^{(l)}$, respectively. For some n-dimensional real random vectors $\mathbf{x}$ and $\mathbf{y}$ defined over finite alphabets $\mathcal{X}^n$ and Ω^n, we denote entropy of $\mathbf{x}$ by $\mathcal{H}(\mathbf{x})$ and mutual information between $\mathbf{x}$ and $\mathbf{y}$ by $\mathcal{I}(\mathbf{x};\mathbf{y})$. Moreover, information loss is defined by $\mathcal{L}(\mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x}|\mathbf{y}^{(l)})$, i.e., relative entropy of $\mathbf{x}$ with respect to $\mathbf{y}^{(l)}$. The characteristics of the layer-wise propagation rule of GCN lead us to the following result:

Lemma 1 For GCNs with parametric ReLU activations $\sigma : x \to \max(x, ax)$ with $a \in (0,1)$, let $\mathbf{P}^{(i+1)}$ be a diagonal matrix whose nonzero entries are in $\{a, 1\}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\big((\mathbf{W}^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\big)_j \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ elsewhere. $\mathbf{y}^{(l)}$ can be written as

$$\mathbf{y}^{(l)} = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})\mathbf{x}.$$

Further, our theory predicts the information transferred across the network exponentially decays to zero as follows.

Theorem 1 Let GCN follows the propagation rule introduced earlier. Suppose $\sigma_{\mathbf{A}} = \max_j \lambda_j(\mathbf{A})$ and $\sigma_{\mathbf{W}} = \sup_{i\in\mathbb{N}^+} \max_j \lambda_j(\mathbf{W}^{(i)})$. If $\sigma_{\mathbf{A}}\sigma_{\mathbf{W}} < 1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = \mathcal{O}\big((\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l\big)$, and hence $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$. In particular Theorem 1 also holds for traditional ReLU with $f : x \to x^+ = \max(0, x)$.

There are also regimes in which GCN will perfectly preserve the information, stated as follows:

Theorem 2 Following Theorem 1, let now $\gamma_{\mathbf{A}} = \min_j \lambda_j(\mathbf{A})$ and $\gamma_{\mathbf{W}} = \inf_{i\in\mathbb{N}^+} \min_j \lambda_j(\mathbf{W}^{(i)})$. If $a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}} \geq 1$, then $\forall l \in \mathbb{N}^+ \; \mathcal{L}(\mathbf{y}^{(l)}) = 0$.

Effect of Normalized Laplacian: The results obtained above holds for any adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. The unnormalized $\mathbf{A}$, however, comes with a major drawback as changing the scaling of feature vectors. To overcome this problem, $\mathbf{A}$ is often normalized such that its rows sum to one. We then adopt our results to GCN with normalized Laplacian whose largest singular value is one. We have the following result:

Corollary 1 Let $\mathbf{D}$ denote the degree matrix such that $(\mathbf{D})_{j,j} = \sum_m (\mathbf{A})_{j,m}$, and $\mathbf{L}$ be the associated normalized Laplacian $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Suppose GCN uses the following mapping $\mathbf{Y}^{(i+1)} = \sigma(\mathbf{L}\mathbf{Y}^{(i)}\mathbf{W}^{(i)})$. Let also $\sigma_{\mathbf{W}} = \sup_i \max_j \lambda_j(\mathbf{W}^{(i+1)})$. If $\sigma_{\mathbf{W}} < 1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = \mathcal{O}\big(\sigma_{\mathbf{W}}^l\big)$, and hence $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.

GraphCNN

Similarly as in Lemma 1, $\mathbf{y}^{(l)}$ can be reduced to $\mathbf{y}^{(l)} = \mathbf{P}^{(l)} \sum_{\mathbf{k_l}=\mathbf{1}}^{\mathbf{K}} (\mathbf{W}_{\mathbf{k_l}}^{(\mathbf{l})} \otimes \mathbf{A}_{\mathbf{k_l}}) \cdots (\mathbf{W}_{\mathbf{k_2}}^{(\mathbf{2})} \otimes \mathbf{A}_{\mathbf{k_2}})(\mathbf{W}_{\mathbf{k_1}}^{(\mathbf{1})} \otimes \mathbf{A}_{\mathbf{k_1}})\mathbf{x}$ for a diagonal matrix $\mathbf{P}^{(i+1)}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\sum_{k_{i+1}=1}^{K} (\mathbf{W}_{k_{i+1}}^{(i+1)} \otimes \mathbf{A}_{k_{i+1}})\mathbf{y}^{(i)} \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ otherwise. We obtain the following result for GraphCNN:

Theorem 3 Let $\sigma^{(i)}$ denotes the maximum singular value of $\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})$ such that $\sigma^{(i)} = \max_j \lambda_j\big(\mathbf{P}^{(i)} \sum_{k_i} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})\big)$. If $\sup_{i\in\mathbb{N}^+} \sigma^{(i)} < 1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = \mathcal{O}\big((\sup_{i\in\mathbb{N}^+} \sigma^{(i)})^l\big)$, and hence $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$. Theorem 3 describes the condition on the layer-wise weight matrices $\mathbf{W}_k$ where GraphCNN fails in capturing the feature characteristics at its output in the asymptotic regime. We then state the second result for GraphCNN which ensures the information loss $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ as follows.

Theorem 4 Consider the propagation rule of GraphCNN. Let $\gamma^{(i)}$ denotes the minimum singular value of $\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})$ such that $\gamma^{(i)} = \min_j \lambda_j\big(\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})\big)$. If $\inf_i \sigma^{(i)} \geq 1$, then $\forall l \in \mathbb{N}^+$ we have $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

In order to understand the role of decomposition in GraphCNN, we revisit the conditions on full information loss ($\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$) and full information preservation ($\mathcal{L}(\mathbf{y}^{(l)}) = 0$) for a specific choice of decomposition, which will later be used to demonstrate the information processing capability of GraphCNN.

Corollary 2 Suppose the singular value decomposition of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{U}_{\mathbf{A}}\mathbf{S}\mathbf{V}_{\mathbf{A}}^T$, and each $\mathbf{A}_k$ is set to $\mathbf{A}_k = \mathbf{U}_{\mathbf{A}}\mathbf{S}_k\mathbf{V}_{\mathbf{A}}^T$ where $(\mathbf{S}_k)_{m,m} = \lambda_m(\mathbf{A})$ if $k = m$ and $(\mathbf{S}_k)_{m,m} = 0$ elsewhere. We then have the following results: For $\sigma_{\mathbf{A}_k} = \lambda_k(\mathbf{A})$ and $\sigma_{\mathbf{W}_k} = \sup_{i\in\mathbb{N}^+} \max_j \lambda_j(\mathbf{W}_k^{(i)})$, i.e., if $\sigma_{\mathbf{A}_k}\sigma_{\mathbf{W}_k} < 1$ $\underline{\forall k = \{1, 2, \ldots, n\}}$, then $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.

Corollary 3 Let $\gamma_{\mathbf{W}_k} = \inf_{i\in\mathbb{N}^+} \min_j \lambda_j(\mathbf{W}_k^{(i)})$. If $a\sigma_{\mathbf{A}_k}\gamma_{\mathbf{W}_k} \geq 1$, $\underline{\forall k \in \{1, 2, \ldots, n\}}$, then $\mathcal{L}(\mathbf{y}^{(l)}) = 0 \; \forall l \in \mathbb{N}^+$. While the universally optimal decomposition strategy is unknown and its existence is debatable, the choice of decomposition introduced above will later highlight the dramatic difference between the capabilities of GCN and GraphCNN.

Discussion: GCN vs. GraphCNN

Consider the setting where $\mathbf{A}$ is fixed and same for both GCN and GraphCNN. The discussions below will revolve around the regime of singular values of layer-wise weight matrices, $\mathbf{W}_{\text{GCN}}^{(i)}$ and $\mathbf{W}_{\text{GraphCNN}}^{(i)}$ where the information loss $\mathcal{L}(\mathbf{y}^{(l)}) = 0$, *for the specific decomposition strategy used in Corollary 3.* Recall from Theorem 2 and Corollary 2 that while GCN requires singular values of all weight matrices $\mathbf{W}_{\text{GCN}}^{(i)}$ to compensate for the minimum singular value of $\mathbf{A}$ such that $\min_j \lambda_j(\mathbf{W}_{\text{GCN}}^{(i)}) \geq \frac{1}{a \min_k \lambda_k(\mathbf{A})}$ to ensure $\mathcal{L}(\mathbf{y}^{(l)}) = 0$, GraphCNN relaxes this condition by introducing a milder constraint. That is, the singular values of its weight matrices $\mathbf{W}_{\text{k, GraphCNN}}^{(i)}$ need to compensate only for the singular value of their respective component $\mathbf{A}_k$, that is, $\min_j \lambda_j(\mathbf{W}_{\text{k, GraphCNN}}^{(i)}) \geq \frac{1}{a\lambda_k(\mathbf{A})}$ guarantees that $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. In other words, singular values of weight matrices of GraphCNN are lower bounded by much smaller values than that of GCN such that information can be fully recovered at the output layer, hence $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ yields for GraphCNN in a much larger regime of weights, hence GraphCNN is better capable of going deeper than GCN by preserving more information about the node features at its output.

References

Beck, D.; Haffari, G.; and Cohn, T. 2018. Graph-to-sequence learning using gated graph neural networks. *arXiv preprint arXiv:1806.09835*.

Belghazi, M. I.; Baratin, A.; Rajeshwar, S.; Ozair, S.; Bengio, Y.; Courville, A.; and Hjelm, D. 2018. Mutual information neural estimation. In Dy, J., and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 531–540. Stockholmsmssan, Stockholm Sweden: PMLR.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13(Feb):281–305.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Janocha, K., and Czarnecki, W. M. 2017. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

Kipf, T. N., and Welling, M. 2016a. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kipf, T. N., and Welling, M. 2016b. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Li, Y.; Yu, R.; Shahabi, C.; and Liu, Y. 2017. Graph convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

Li, G.; Müller, M.; Thabet, A.; and Ghanem, B. 2019. Can gcns go as deep as cnns? *arXiv preprint arXiv:1904.03751*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Oono, K., and Suzuki, T. 2019. On asymptotic behaviors of graph cnns from dynamical systems perspective. *arXiv preprint arXiv:1905.10947*.

Rahimi, A.; Cohn, T.; and Baldwin, T. 2018. Semi-supervised user geolocation via graph convolutional networks. *arXiv preprint arXiv:1804.08049*.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Such, F. P.; Sah, S.; Dominguez, M. A.; Pillai, S.; Zhang, C.; Michael, A.; Cahill, N. D.; and Ptucha, R. 2017. Robust spatial filtering with graph convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing* 11(6):884–896.

Telatar, E. 1999. Capacity of multiantenna gaussian channels. *European transactions on telecommunications* 10:585–595.

Wu, F.; Zhang, T.; Souza Jr, A. H. d.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019a. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019b. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Xu, D.; Zhu, Y.; Choy, C. B.; and Fei-Fei, L. 2017. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5410–5419.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Supplementary Material

A. Detailed Model Architectures

We compare three model architectures: CNN, GCN, and GraphCNN:

1. CNN (Krizhevsky, Sutskever, and Hinton, 2012) The architecture is stacked by 3×3 convolution layers. The input channel of the first CNN layer is 3 (including RGB) and the output channel is set as 128. All the input and output channels of the succeeding convolution layers are 128.

2. GCN (Kipf and Welling, 2016b) We treat all edges in the graph equally and leverage a similar network architecture as CNN. The only difference is that we replace each 3×3 convolution layer with a GCN layer.

3. GraphCNN (Such et al., 2017) We replace each convolution layer with a GraphCNN layer, which is decomposed as illustrated in Figure 4. Specifically, we decompose the adjancency matrix $\mathbf{A}$ into 9 submatrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_9$. For two arbitrary pixels (i, j) and (m, n), we set the edges $e(i, j, m, n) = 1$ of each submatrix $\mathbf{A}_i$ when the following equation holds; otherwise the corresponding edges are set as zero in that matrix: (1) $i = j$ and $m = n$; (2) $i + 1 = j$ and $m = n$; (3) $i = j + 1$ and $m = n$; (4) $i = j$ and $m + 1 = n$; (5) $i = j$ and $m = n + 1$; (6) $i + 1 = j$ and $m + 1 = n$; (7) $i + 1 =$ and $m = n + 1$; (8) $i = j + 1$ and $m + 1 = n$; (9) $i = j + 1$ and $m = n + 1$.

For each of these three architectures, we focus on the following techniques:

1. Original. Applying (graph) convolution operations in each layer with $stride = 1$ and with no skip of connections, we reshape the 2D image to a 1D embedding vector and add a fully connected layer with Softmax activation on the top to generate classification results at the last layer, where all hidden size is set to 128.

2. Stride. The stride of each layer is aligned with ResNet-18. Specifically for the 9th, 13th, and 17th layer, we apply $stride = 2$, and both the length and width of the original image will be halved. We follow a common strategy with the hidden size is doubled upon a stride operation (original hidden size = 128). To imitate the stride behavior for GCN and GraphCNN, we perform convolution first, then choose the nodes that will be reserved by the strides to construct a new grid graph corresponding to the smaller image.

3. Stride+Skip. We add skip connections between the corresponding layers (i.e., $1^{st} \rightarrow 3^{rd}$, $3^{rd} \rightarrow 5^{th}$, $5^{th} \rightarrow 7^{th}$, $7^{rd} \rightarrow 9^{th}$, $9^{rd} \rightarrow 11^{th}$, $11^{st} \rightarrow 13^{rd}$, $13^{rd} \rightarrow 15^{th}$, $15^{rd} \rightarrow 17^{th}$) following the standard architecture of ResNet-18 (See XXX for more details). Other configurations are kept the same with the **Stride** setting.

4. Stride+Skip+AP. The network architecture of the 17-layer CNN looks similar to ResNet-18 except that it does not adopt average pooling before the final fully connected layer. To align with ResNet-18, we also compare the models in the architectures with average pooling on the top such that the 17-layer CNN exactly matches the network architecture of a standard ResNet-18.

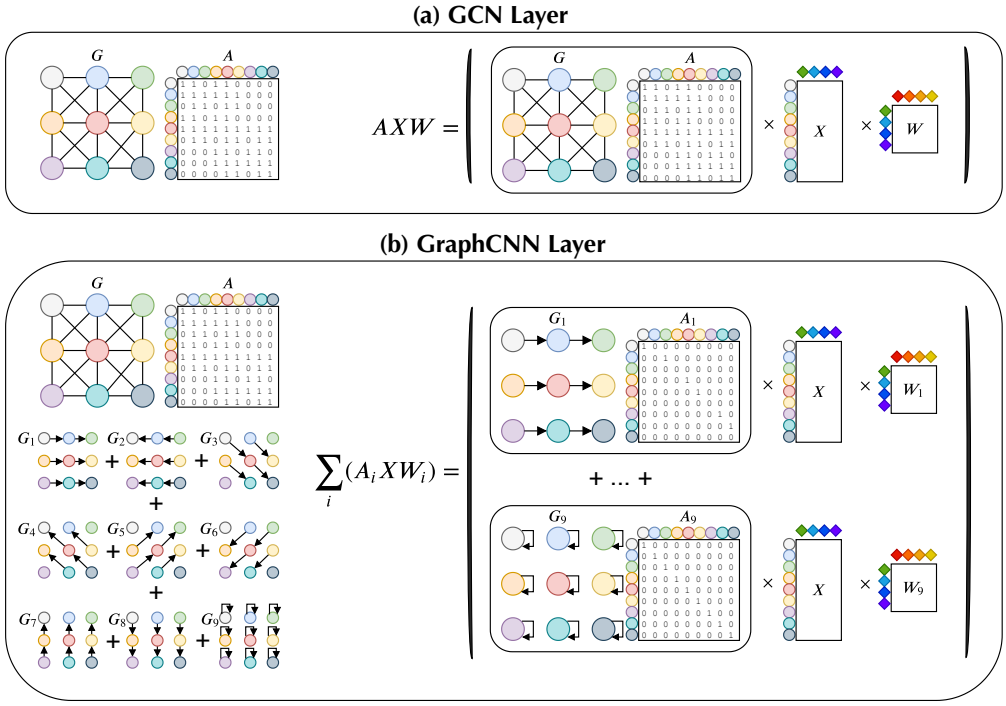


Figure 4: Illustration of one layer in GCN and one layer under one decomposition strategy in GraphCNN. $\mathbf{A}$ is the adjacency matrix, $\mathbf{X}$ is the input, and $\mathbf{W}$ ($\mathbf{W}_i$) are learnable weights. In GraphCNN, $\mathbf{A} = \sum_i \mathbf{A}_i$ and $\mathbf{A}_i \cap \mathbf{A}_j = \emptyset$ for $i \neq j$. In our experiments and analysis, we follow the original paper and normalize $\mathbf{A}$ in GCN.

B. Proofs

We begin by introducing our notation. Hereafter, scalars will be written in italics, vectors in bold lower-case and matrices in bold upper-case letters. For an $m \times n$ real matrix $\mathbf{A}$, the matrix element in the ith row and jth column is denoted as $(\mathbf{A})_{ij}$, and ith entry of a vector $\mathbf{a} \in \mathbb{R}^m$ by $(\mathbf{a})_i$. Also, jth column of $\mathbf{A}$ is denoted by $(\mathbf{A})_j$, or $(\mathbf{A})_{[i=1,2,\ldots,m],j}$. Similarly, we denote ith row by $(\mathbf{A})_{i,[j=1,2,\ldots,n]}$. The inner product between two vectors $(\mathbf{A})_i$ and $(\mathbf{A})_{i'}$ is denoted by $\langle (\mathbf{A})_i, (\mathbf{A})_{i'} \rangle$.

In the next section, we will first introduce the outline of proofs.

Outline of the Proof: Following Lemma 1, the next key step in proving above results is as follows.

Lemma 2 Consider the singular value decomposition $\mathbf{U}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ such that $(\mathbf{\Lambda})_{j,j} = \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A}))$, and let $\tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$. We have

$$\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) \overset{(1)}{=} \mathcal{I}(\tilde{\mathbf{x}}; \Lambda\tilde{\mathbf{x}}) \overset{(2)}{\leq} \mathcal{H}(\tilde{\mathbf{x}}) \overset{(3)}{=} \mathcal{H}(\mathbf{x}) \tag{1}$$

where (1, 3) results from that $\mathbf{U}$ and $\mathbf{V}$ are invertible, and equality holds in (2) *iff* $\mathbf{\Lambda}$ is invertible, i.e., singular values of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ are nonzero.

Theorem 1, 2, 3 and 4 can easily be inferred from Lemma 2. That is, $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ *iff* $\max_j(\mathbf{\Lambda}^l)_{j,j} = 0$ in the asymptotic regime. Similarly, *iff* $\min_j(\mathbf{\Lambda}^l)_{j,j} > 0$, $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)})$ is maximized and given by $\mathcal{H}(\mathbf{x})$, hence $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

Our results presented so far focus on covering the edge cases: $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ or $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. While our primary goal is to understand why GraphCNN has a better capability of going deep than that of GCN, we note several points about Lemma 2 in a viewpoint of entropy or uncertainty:

1. Rigorous theoretical guarantees quantifying the amount of information preserved across the network is not straightforward, and further requires the knowledge on the statistical properties of node features. Despite its simplicity, Lemma 2 forms a direct link from the information processing capability of the network to the characteristics of the weights and entropy of the nodes, $\mathbf{x}_i$,
2. Whereas the compression and generalization capability of the network are closely related, we emphasize here that our analysis here is to understand why and when GraphCNN overcome the *overcompression* introduced by GCN. In future, we plan to investigate this via the information bottleneck principle,
3. In our formulation, we omit the effect of perturbation in the input nodes considering our discussion will remain valid under the same perturbation characteristics,
4. If all node features $\mathbf{x}_i$, for instance, have similar entropy, $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)})$ roughly linearly scales with the rank of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$,
5. Lifting up singular values of layer-wise weight matrices are beneficial for better data processing in a viewpoint of information theory. In the next section, we will demonstrate through edge cases how GraphCNN can overcome *overcompression* of GCN by achieving singular value lifting.

Proofs: We vectorize a matrix $\mathbf{A}$ by concatenating its columns such that

$$\mathrm{vec}(\mathbf{A}) = \begin{bmatrix} (\mathbf{A})_1 \\ (\mathbf{A})_2 \\ \vdots \\ (\mathbf{A})_n \end{bmatrix}$$

and denote it by $\mathrm{vec}(\mathbf{A})$. For matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{k \times l}$, we denote the kronecker product of $\mathbf{A}$ and $\mathbf{B}$ by $\mathbf{A} \otimes \mathbf{B}$ such that

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} (\mathbf{A})_{11}\mathbf{B} & \ldots & (\mathbf{A})_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ (\mathbf{A})_{m1}\mathbf{B} & \ldots & (\mathbf{A})_{mn}\mathbf{B} \end{bmatrix}.$$

Note that $\mathbf{A} \otimes \mathbf{B}$ is of size $mk \times nl$.

We moreover denote the floor function and modulo operation by $\lfloor \rfloor and \mod$, respectively. Finally, we denote the jth largest singular value of a matrix $\mathbf{A}$ by $\lambda_j(\mathbf{A})$.

Next, we list some existing results which we require repeatedly throughout this section.

Preliminaries.

1. Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times k}$ and $\mathbf{C} \in \mathbb{R}^{k\times p}$. We have
$$\text{vec}(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A})\,\text{vec}(\mathbf{B}). \tag{2}$$
2. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times k}$ and $\mathbf{C} \in \mathbb{R}^{m'\times n'}$, $\mathbf{D} \in \mathbb{R}^{n'\times k'}$
$$(\mathbf{AB} \otimes \mathbf{CD}) = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \otimes \mathbf{D}). \tag{3}$$
3. For $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$, singular values of $\mathbf{A} \otimes \mathbf{B}$ is given by $\lambda_i(\mathbf{A})\lambda_j(\mathbf{B})$, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.
4. Let $\mathbf{x}$ and $\mathbf{y}$ be an n-dimensional random vector defined over finite alphabets $\mathcal{X}^n$ and Ω^n, respectively. We denote entropy of $\mathbf{x}$ by $\mathcal{H}(\mathbf{x})$ and mutual information between $\mathbf{x}$ and $\mathbf{y}$ by $\mathcal{I}(\mathbf{x};\mathbf{y})$. We list the followings:
$$\begin{aligned} \mathcal{H}(f(\mathbf{x})) &\overset{(a)}{\leq} \mathcal{H}(\mathbf{x}) \\ \mathcal{I}(\mathbf{x}; f(\mathbf{y})) &\overset{(b)}{\leq} \mathcal{I}(\mathbf{x};\mathbf{y}) \end{aligned} \tag{4}$$
such that $f : \mathbb{R} \to \mathbb{R}$ is some deterministic function, and equality holds for both inequalities *iff* f is bijective.

Proofs. The proofs are listed below in order.

Proof of Lemma 1. Applying vectorization to the GCN layer-wise propagation rule introduced earlier, we have
$$\begin{aligned} \mathbf{y}^{(i+1)} &= \text{vec}\left(\sigma(\mathbf{AY}^{(i)}\mathbf{W}^{(i+1)})\right) \\ \mathbf{y}^{(i+1)} &\overset{(a)}{=} \sigma\big(\text{vec}(\mathbf{AY}^{(i)}\mathbf{W}^{(i+1)})\big) \\ \mathbf{y}^{(i+1)} &\overset{(b)}{=} \sigma\big(((\mathbf{W}^{(i+1)})^T \otimes \mathbf{A})\mathbf{y}^{(i)}\big) \\ \mathbf{y}^{(i+1)} &\overset{(c)}{=} \mathbf{P}^{(i+1)}((\mathbf{W}^{(i+1)})^T \otimes \mathbf{A})\mathbf{y}^{(i)} \end{aligned} \tag{5}$$
where (a) follows from the element-wise application of σ, (b) follows from (2), and (c) results from introducing a diagonal matrix $\mathbf{P}^{(i+1)}$ with diagonal entries in $\{a, 1\}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\left((\mathbf{W}^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\right)_j \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ elsewhere.

By a recursive application of (5c), we have
$$\mathbf{y}^{(l)} = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \ldots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})\mathbf{x}.$$
□

We drop the transpose from $\mathbf{W}^{(i+1)}$ in order to avoid cumbersome notation. The singular values of $\mathbf{W}^{(i+1)}$ are our primary interest thereof our results still hold.

Proof of Lemma 2. Let $\mathbf{\Sigma}$ be a $n \times n$ matrix with singular value decomposition $\mathbf{\Sigma} = \mathbf{U\Lambda V}^T$. Inspired by the derivation for the capacity of deterministic channels introduced by Telatar (1999), we derive the following
$$\begin{aligned} \mathcal{I}(\mathbf{x}; \mathbf{\Sigma x}) = \mathcal{I}(\mathbf{x}; \mathbf{U\Lambda V}^T\mathbf{x}) &\overset{(a)}{=} \mathcal{I}(\mathbf{x}; \mathbf{\Lambda V}^T\mathbf{x}) \\ \mathcal{I}(\mathbf{x}; \mathbf{\Sigma x}) &\overset{(b)}{=} \mathcal{I}(\mathbf{V}^T\mathbf{x}; \mathbf{\Lambda V}^T\mathbf{x}) \overset{(c)}{=} \mathcal{I}(\tilde{\mathbf{x}}; \mathbf{\Lambda}\tilde{\mathbf{x}}). \end{aligned} \tag{6}$$
(a) and (b) are a result of (4b) and that $\mathbf{U}$ and $\mathbf{V}$ are unitary hence invertible (bijective) transformations. (c) follows from the change of variables $\tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$.

Note that $\mathcal{I}(\tilde{\mathbf{x}}; \mathbf{\Lambda}\tilde{\mathbf{x}}) \leq \mathcal{H}(\mathbf{\Lambda}\tilde{\mathbf{x}})$. Using (4a), we further have $\mathcal{H}(\mathbf{\Lambda}\tilde{\mathbf{y}}) \leq \mathcal{H}(\tilde{\mathbf{x}}) = \mathcal{H}(\mathbf{x})$ which completes the proof. □

We recall that we are interested in regimes where $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ and $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. In Lemma 2, we show that $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ if $\max_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) = 0$, and maximized (and given by $\mathcal{H}(\mathbf{x})$) when $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ is invertible. Therefore, maximum and minimum singular values of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ are of our interest.

Proof of Theorem 1. Let $\sigma_{\mathbf{A}} = \max_j \lambda_j(\mathbf{A})$ and $\sigma_{\mathbf{W}} = \sup_i \max_j \lambda_j(\mathbf{W}^{(i)})$. That is, given singular values of $\mathbf{P}^{(i)}$ is in $\{a, 1\}$, $\sup_i \max_j \lambda_j(\mathbf{P}^{(i)}(\mathbf{W}^{(i)} \otimes \mathbf{A})) = \sigma_{\mathbf{A}}\sigma_{\mathbf{W}}$. We, moreover, have $\max_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) \leq (\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l$. Therefore, if $\sigma_{\mathbf{A}}\sigma_{\mathbf{W}} < 1$, by Lemma 2 we have $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = \mathcal{O}((\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l)$, and $\lim_{l\to\infty} \mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$. □

Proof of Theorem 2. We now denote $\gamma_{\mathbf{A}} = \min_j \lambda_j(\mathbf{A})$ and $\gamma_{\mathbf{W}} = \inf_i \min_j \lambda_j(\mathbf{W}^{(i)})$. Hence $\inf_i \min_j \lambda_j(\mathbf{P}^{(i)}(\mathbf{W}^{(i)} \otimes \mathbf{A})) = a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}}$. Moreover, $\min_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) \geq (a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}})^l$. If $a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}} \geq 1$, $\min_j \lambda_j(\mathbf{P}_l(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}_2(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}_1(\mathbf{W}^{(1)} \otimes \mathbf{A})) \geq 1 \; \forall l \in \mathbb{N}^+$, hence $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x})$ and $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ results by Lemma 2. □

Proof of Corollary 1. Let $\mathbf{D}$ denote the degree matrix such that $(\mathbf{D})_{j,j} = \sum_m (\mathbf{A})_{j,m}$, and $\mathbf{L}$ be the associated normalized Laplacian $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Due to the property of normalized Laplacian such that $\max_j \lambda_j(\mathbf{L}) = 1$, we have $\sigma_{\mathbf{A}} = 1$. Inserting this into Theorem 1, the corollary results. □

Similarly as in (5), $\mathbf{y}^{(i+1)}$ can be derived from the single layer response of GraphCNN as follows:

$$\begin{aligned} \mathbf{y}^{(i+1)} &= \operatorname{vec}\left(\sigma(\sum_k \mathbf{A}_k \mathbf{Y}^{(i)} \mathbf{W}_k^{(i+1)})\right) \overset{(a)}{=} \sigma(\sum_k \operatorname{vec}(\mathbf{A}_k \mathbf{Y}^{(i)} \mathbf{W}_k^{(i+1)})) \\ \mathbf{y}^{(i+1)} &\overset{(b)}{=} \sigma(\sum_k (\mathbf{W}_k^{(i+1)} \otimes \mathbf{A}_k)\mathbf{y}^{(i)}\sigma) \overset{(c)}{=} \mathbf{P}^{(i+1)} \sum_k (\mathbf{W}_k^{(i+1)} \otimes \mathbf{A}_k)\mathbf{y}^{(i)} \end{aligned} \tag{7}$$

where $\mathbf{P}^{(i+1)}$ is a diagonal matrix with diagonal entries in $\{a, 1\}$ with $a \in (0, 1)$ such that $(\mathbf{P}^{(i)})_{j,j} = 1$ if $\left(\sum_k (\mathbf{W}_k^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\right)_j \geq 0$, and $(\mathbf{P}^{(i)})_{j,j} = a$ otherwise.

Therefore, $\mathbf{y}^{(l)}$ is given by

$$\mathbf{y}^{(l)} = \mathbf{P}^{(l)} \sum_{k_l} (\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots \mathbf{P}^{(2)} \sum_{k_2} (\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2})\mathbf{P}^{(1)} \sum_{k_1} (\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})\mathbf{x}.$$

Consider (6) where $\mathbf{\Sigma}$ is replaced with $\mathbf{P}^{(l)} \sum_{k_l} (\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots \mathbf{P}^{(2)} \sum_{k_2} (\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2})\mathbf{P}^{(1)} \sum_{k_1} (\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})$.
We deduce the followings:

Proof of Theorem 3. Suppose $\sigma^{(i)}$ denotes the largest singular value of $\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})$ such that $\sigma^{(i)} = \max_j \lambda_j\left(\mathbf{P}^{(i)} \sum_{k_i} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})\right)$. Following the same argument as in the proofs of Theorem 1 and 2, Lemma 2 implies that if $\sup_i \sigma^{(i)} < 1$, then $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = \mathcal{O}\left((\sup_i \sigma^{(i)})^l\right)$, and hence $\lim_{l \to \infty} \mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ results. □

Proof of Theorem 4. We now $\gamma^{(i)}$ denote the minimum singular value of $\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})$ such that $\gamma^{(i)} = \min_j \lambda_j\left(\mathbf{P}^{(i)} \sum_{k_i=1}^{K} (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})\right)$. By Lemma 2, it immediately follows that if $\inf_i \sigma^{(i)} \geq 1$, then $\forall l \in \mathbb{N}^+$ we have $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. □

Before we move on to the proofs of Corollary 2 and 3, we state the following lemma.

Lemma 3 Let the singular value decomposition of $\mathbf{A} \in \mathbb{R}^{n \times n}$ is given by $\mathbf{A} = \mathbf{U}_{\mathbf{A}}\mathbf{S}\mathbf{V}_{\mathbf{A}}^T$ and we set each $\mathbf{A}_k$ to $\mathbf{A}_k = \mathbf{U}_{\mathbf{A}}\mathbf{S}_k\mathbf{V}_{\mathbf{A}}^T$ with $(\mathbf{S}_k)_{m,m} = \lambda_m(\mathbf{A})$ if $k = m$ and $(\mathbf{S}_k)_{m,m} = 0$ elsewhere. For such specific composition, we argue that singular values of $\sum_k \mathbf{W}_k \otimes \mathbf{A}_k$ for $\mathbf{W}_k \in \mathbb{R}^{d \times d}$ is given by $\lambda_k(\mathbf{A})\lambda_j(\mathbf{W}_k)$ for $k = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, d$.

Proof of Lemma 3. Let the singular value decomposition of $\mathbf{W}_k$ be $\mathbf{W}_k = \mathbf{U}_{\mathbf{W}_k}\mathbf{S}_{\mathbf{W}_k}\mathbf{V}_{\mathbf{W}_k}^T$. By the property of kronecker product, we have

$$\sum_k \mathbf{W}_k \otimes \mathbf{A}_k = \sum_k (\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)(\mathbf{V}_{\mathbf{W}_k}^T \otimes \mathbf{V}_{\mathbf{A}}^T).$$

Next, we define a set of $nd \times nd$ mask matrices $\mathbf{M}_k$ such that $(\mathbf{M}_k)_{i,i'} = 1$ if $i = i'$ and i (hence i') is of the form $i = k + (j-1)n$ for $j = 1, 2, \ldots, d$, and $(\mathbf{M}_k)_{i,i'} = 0$ otherwise. Reminding that $(\mathbf{S}_k)_{m,m} = \lambda_m(\mathbf{A})$ if $k = m$ and $(\mathbf{S}_k)_{m,m} = 0$ elsewhere, above equation can be rewritten as

$$\sum_k \mathbf{W}_k \otimes \mathbf{A}_k = \sum_k (\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)\mathbf{M}_k(\mathbf{V}_{\mathbf{W}_k}^T \otimes \mathbf{V}_{\mathbf{A}}^T).$$

In other words, the mask matrix $\mathbf{M}_k$ applies on the columns (rows) of $\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}}$ $(\mathbf{V}_{\mathbf{W}_k}^T \otimes \mathbf{V}_{\mathbf{A}}^T)$ where the respective diagonal entries of $(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)$ are nonzero.

Next, we note that if $k = k'$, $\mathbf{M}_k\mathbf{M}_{k'} = \mathbf{M}_k$, and $\mathbf{M}_k$ and $\mathbf{M}_{k'}$ are orthogonal for $k \neq k'$. This leads us to

$$(\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)\mathbf{M}_k(\mathbf{V}_{\mathbf{W}_k}^T \otimes \mathbf{V}_{\mathbf{A}}^T) = \sum_{k'} (\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k) \sum_{k''} (\mathbf{V}_{\mathbf{W}_{k''}}^T \otimes \mathbf{V}_{\mathbf{A}}^T)\mathbf{M}_{k''}.$$

By defining $\tilde{\mathbf{U}} = \sum_k (\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k$ and $\tilde{\mathbf{V}} = \sum_k \mathbf{M}_k(\mathbf{V}^T_{\mathbf{W}_k} \otimes \mathbf{V}^T_{\mathbf{A}})$ and using the above equation, we resume $\sum_k \mathbf{W}_k \otimes \mathbf{A}_k$ as

$$\sum_k \mathbf{W}_k \otimes \mathbf{A}_k = \tilde{\mathbf{U}} \sum_k (\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)\tilde{\mathbf{V}}^T. \tag{8}$$

Next, we will show that $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are unitary matrices through proving that $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T = \tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \mathbf{I}$ and $\tilde{\mathbf{V}}^T\tilde{\mathbf{V}} = \tilde{\mathbf{V}}\tilde{\mathbf{V}}^T = \mathbf{I}$. To avoid repeating the same procedure, we will only show it for $\tilde{\mathbf{U}}$, but the same result also holds for $\tilde{\mathbf{V}}$.

First, we show that **(A.1)** $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T = \mathbf{I}$, and then **(A.2)** $\tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \mathbf{I}$ to argue that $\tilde{\mathbf{U}}$ (and $\tilde{\mathbf{V}}$) is unitary.

(A.1) We can simplify $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T$ as

$$\begin{aligned}
\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T &= \sum_k \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big) \sum_{k'} \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)^T \\
\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T &= \sum_{k,k'} \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)^T \\
\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T &\overset{(a)}{=} \sum_k \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T
\end{aligned} \tag{9}$$

where (a) follows from the orthogonality of $\mathbf{M}_k$ and $\mathbf{M}_{k'}$ for $k \neq k'$.

We will now take a closer look at $\sum_k \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T$. The entries of summands, $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T$, are equivalent to inner product between the rows of $(\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k$ for a fixed k. Recall that for a fixed k, the mask matrix satisfies $(\mathbf{M}_k)_{i,i} = 1$ if k is of the form $i = k + (j-1)n$ for $j = 1, 2, \cdots, d$, and $(\mathbf{M}_k)_{i,i} = 0$ elsewhere. We now define i_ω and i_α as indices such that $i_\omega = \lfloor i/n \rfloor + 1$ and $i_\alpha = \mod(i, \lfloor i/n \rfloor)$. Similarly, let $i'_\omega = \lfloor i'/n \rfloor + 1$ and $i'_\alpha = \mod(i', \lfloor i'/n \rfloor)$.

Following above definitions, a moment of thought reveals that the nonzero entries of ith row of $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)$ is given by $(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}$. We therefore investigate $(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i'}$ i.e., the inner product between ith and i'th rows of $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)$ summed over all $k = 1, 2, \ldots, n$. To start, the inner product between ith and i'th rows of $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)$ is as follows

$$\begin{aligned}
&\langle [(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}], [(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}] \rangle = \sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k} \\
&= \sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k} = (\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k} \sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}.
\end{aligned} \tag{10}$$

Let now analyze the cases when (1) $i \neq i'$, and (2) $i = i'$.

Assume (1). If further $i_\omega \neq i'_\omega$, it is immediate that $\sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m} = 0$ by the fact that $\mathbf{U}_{\mathbf{W}_k}$ is unitary, hence

$$\langle [(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}], [(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}] \rangle = 0$$

For (1), if $i_\omega = i'_\omega$, we have $i_\alpha \neq i'_\alpha$. Further, $\sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m} = 1$ and hence

$$\begin{aligned}
\langle [(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}], [(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,[m=1,2,\ldots,d]}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}] \rangle &= (\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k} \sum_m (\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m} \\
&= (\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}.
\end{aligned} \tag{11}$$

Hence, the inner product between ith and i'th rows of $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)$ is given by $(\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}$. Recalling (9), we have $(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i'} = \sum_k (\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k}$. As previously mentioned we have $i_\alpha \neq i'_\alpha$. By the unitary property of $\mathbf{U}_{\mathbf{A}}$, we further have $(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i'} = \sum_k (\mathbf{U}_{\mathbf{A}})_{i_\alpha,k}(\mathbf{U}_{\mathbf{A}})_{i'_\alpha,k} = 0$.

So far we have shown that $(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i'} = 0$ when $i \neq i'$. Let now $i = i'$, i.e., (2). IT follows from (10) that

$$(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i} \overset{(a)}{=} \sum_k (\mathbf{U}_{\mathbf{A}})^2_{i_\alpha,k} \sum_m (\mathbf{U}_{\mathbf{W}_k})^2_{i_\omega,m} (\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i} \overset{(b)}{=} \sum_k (\mathbf{U}_{\mathbf{A}})^2_{i_\alpha,k} 1 (\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i} \overset{(c)}{=} 1 \tag{12}$$

where (a) results from that $\mathbf{U}_{\mathbf{W}_k}$ is unitary, and (b) follows from that $\mathbf{U}_{\mathbf{A}}$ is unitary. Combining above arguments and (12), we have $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T = \mathbf{I}$.

(A.2) Next, we show that $\tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \mathbf{I}$. We begin with

$$\tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \sum_k \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T \big(\sum_{k'} (\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)\tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \sum_{k,k'} \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big). \tag{13}$$

For $k \neq k'$,

$$\Big(\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)\Big)_{i,i'} = \langle \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)_i, \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)_{i'} \rangle. \tag{14}$$

Note that, due to the orthogonality of $\mathbf{M}_k$ and $\mathbf{M}_k$ for $k \neq k'$, we further have $\langle \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)_i, \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)_{i'} \rangle = 0$ for $i \neq i'$. When $i = i'$, on the other hand, we have

$$\begin{aligned}
\Big(\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)\Big)_{i,i'} &= \langle \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)_i, \big((\mathbf{U}_{\mathbf{W}_{k'}} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_{k'}\big)_i \rangle \\
&\overset{(a)}{=} \langle (\mathbf{U}_{\mathbf{W}_k})_{[z=1,\cdots,d],i_\omega} (\mathbf{U}_{\mathbf{A}})_{[w=1,\cdots,n],k}, (\mathbf{U}_{\mathbf{W}_{k'}})_{[z=1,\cdots,d],i_\omega} (\mathbf{U}_{\mathbf{A}})_{[w=1,\cdots,n],k'} \rangle \\
&= \sum_w \sum_d (\mathbf{U}_{\mathbf{W}_k})_{z,i_\omega} (\mathbf{U}_{\mathbf{A}})_{w,k} (\mathbf{U}_{\mathbf{W}_{k'}})_{z,i_\omega} (\mathbf{U}_{\mathbf{A}})_{w,k'} \\
&\overset{(b)}{=} \sum_d (\mathbf{U}_{\mathbf{W}_k})_{z,i_\omega} (\mathbf{U}_{\mathbf{W}_{k'}})_{z,i_\omega} \sum_w (\mathbf{U}_{\mathbf{A}})_{w,k} (\mathbf{U}_{\mathbf{A}})_{w,k'} \\
&= 0
\end{aligned} \tag{15}$$

where (a) follows from that $\big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)_i = (\mathbf{U}_{\mathbf{W}_k})_{[z=1,\cdots,d],i_\omega} (\mathbf{U}_{\mathbf{A}})_{[w=1,\cdots,n],k}$ and (b) results from that $\sum_w (\mathbf{U}_{\mathbf{A}})_{w,k} (\mathbf{U}_{\mathbf{A}})_{w,k'} = 0$ for $k \neq k'$ as $\mathbf{U}_{\mathbf{A}}$ is unitary.

Therefore, (13) can be resumed as

$$\begin{aligned}
\tilde{\mathbf{U}}^T \tilde{\mathbf{U}} &= \sum_k \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big)^T \big((\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})\mathbf{M}_k\big) \\
\tilde{\mathbf{U}}^T \tilde{\mathbf{U}} &= \sum_k \mathbf{M}_k (\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})^T (\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}}) \mathbf{M}_k \\
\tilde{\mathbf{U}}^T \tilde{\mathbf{U}} &\overset{(a)}{=} \sum_k \mathbf{M}_k \mathbf{I} \mathbf{M}_{\mathbf{k}} = \sum_{\mathbf{k}} \mathbf{M}_{\mathbf{k}} \overset{(\mathbf{b})}{=} \mathbf{I}
\end{aligned}$$

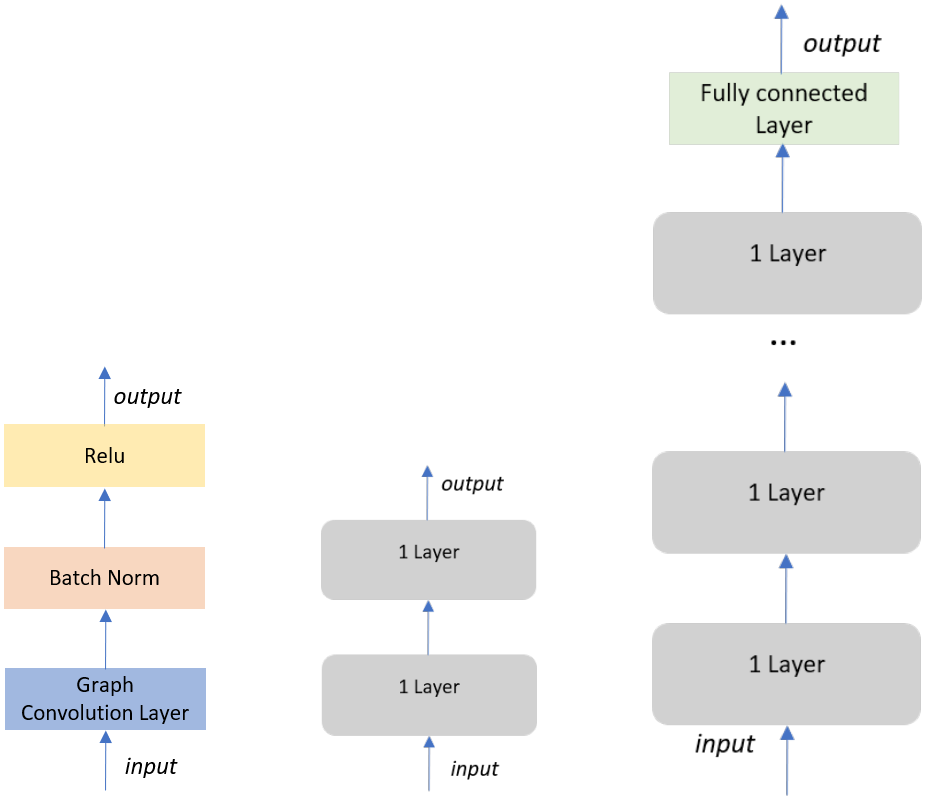
where (a) follows from that the kronecker product of unitary matrices is also unitary, hence $(\mathbf{U}_{\mathbf{W}_k} \otimes \mathbf{U}_{\mathbf{A}})$ is unitary, and (b) follows from the definition of $\mathbf{M}_k$.

As the last step, recall from (8) that $\sum_k \mathbf{W}_k \otimes \mathbf{A}_k = \tilde{\mathbf{U}} \sum_k (\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k) \tilde{\mathbf{V}}^T$, and note by the definition of $\mathbf{S}_k$ that $(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)_{i,i'} = \lambda_k(\mathbf{A}) \lambda_j(\mathbf{S}_{\mathbf{W}_k})$ if $i = i'$ and i, hence i', of the form $i = k + (j-1)n$ for $j = 1, 2, \cdots, d$, and $(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)_{i,i'} = 0$ elsewhere. Therefore, by the fact that $(\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)(\mathbf{S}_{\mathbf{W}_{k'}} \otimes \mathbf{S}_{k'}) = 0$ for $k \neq k'$, it follows that $\sum_k (\mathbf{S}_{\mathbf{W}_k} \otimes \mathbf{S}_k)$ is a diagonal matrix with diagonal entries $\lambda_k(\mathbf{A}) \lambda_j(\mathbf{S}_{\mathbf{W}_k})$ where $j = 1, 2, \cdots, d$ and $k = 1, 2, \cdots, n$, which completes the proof. □

For the decomposition of $\mathbf{A}$ such that $\mathbf{A}_k = \mathbf{U}_{\mathbf{A}} \mathbf{S}_k \mathbf{V}_{\mathbf{A}}^T$ where the singular value decomposition of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{U}_{\mathbf{A}} \mathbf{S} \mathbf{V}_{\mathbf{A}}^T$, we recall Theorem 3 and 4 to conclude Corollary 2 and 3 as follows.

Proof of Corollary 2. Let $\sigma_{\mathbf{A}_k} = \lambda_k(\mathbf{A})$ and $\sigma_{\mathbf{W}_k} = \sup_i \max_j \lambda_j(\mathbf{W}_k^{(i)})$. By Lemma 3, we have $\max_j \lambda_j(\sum_k (\mathbf{W}_k^{(i)} \otimes \mathbf{A}_k)) \leq \max_k \sigma_{\mathbf{A}_k} \sigma_{\mathbf{W}_k}$. Noting that $\mathbf{P}^{(i)}$ is diagonal with entries at most 1, we have $\max_j \lambda_j\big(\mathbf{P}^{(l)} \sum_{k_l} (\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots \mathbf{P}^{(2)} \sum_{k_2} (\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2}) \mathbf{P}^{(1)} \sum_{k_1} (\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})\big) \leq (\max_k \sigma_{\mathbf{A}_k} \sigma_{\mathbf{W}_k})^l$. Therefore, if $\underline{\forall k = \{1, 2, \ldots, n\}}\ \sigma_{\mathbf{A}_k} \sigma_{\mathbf{W}_k} < 1$, then $\lim_{l \to \infty} \max_j \lambda_j\big(\sum_k (\mathbf{W}_k^{(i)} \otimes \mathbf{A}_k)\big) = 0$. Hence $\lim_{l \to \infty} \mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$ results by Lemma 2. □

Proof of Corollary 3. Let $\gamma_{\mathbf{W}_k} = \inf_i \min_j \lambda_j(\mathbf{W}_k^{(i)})$. Note that $\min_j \lambda_j\big(\mathbf{P}^{(i)} \sum_k \mathbf{W}_k^{(i)} \otimes \mathbf{A}_k\big) \geq a \min_k \lambda_k(\mathbf{A}) \gamma_{\mathbf{W}_k}$ by Lemma 3 and that $\min_j \lambda_j(\mathbf{P}^i) = a$. Moreover, $\min_j \lambda_j\big(\mathbf{P}^{(l)} \sum_{k_l} (\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots \mathbf{P}^{(2)} \sum_{k_2} (\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2}) \mathbf{P}^{(1)} \sum_{k_1} (\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})\big) \geq (a \min_k \lambda_k(\mathbf{A}) \gamma_{\mathbf{W}_k})^l$. Therefore, if $a \sigma_{\mathbf{A}_k} \gamma_{\mathbf{W}_k} \geq 1$, $\underline{\forall k \in \{1, 2, \ldots, n\}}$, then $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x})\ \forall l \in \mathbb{N}^+$ by Lemma 2, hence $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. □



(a) 1-Layer (b) 2-Layers (c) 18-Layers

Figure 5: Architecture illustration of different layers

Setting	GCN (*Train*, *Test Acc*)	GraphCNN (*Train*, *Test Acc*)	GraphCNN-random1 (*Train*, *Test Acc*)	GraphCNN-random2 (*Train*, *Test Acc*)	GraphCNN-random3 (*Train*, *Test Acc*)
Original	56.8%, 56.2%	99.1%, 88.6%	69.3%, 67.7%	67.5%, 67.1%	68.3%, 68.0%
Stride	88.9%, 53.1%	99.9%, 93.1%	96.1%, 74.9%	96.8%, 76.3%	97.2%, 75.0%
Stride+Skip	80.0%, 63.5%	100%, 94.5%	98.5%, 83.9%	99.0%, 84.8%	98.8%, 84.1%
Stride+Skip+AP	94.0%, 72.8 %	99.9%, 93.2 %	97.1%, 83.6%	97.4%, 84.4%	96.9%, 83.5%

Table 1: Comparison with randomly decomposed GraphCNN using 17-layer architectures

C. Details of Experiments

The network architectures of GCN/CNN/GraphCNN with different layers are shown in Figure 5; and the decomposition strategy of GraphCNN is illustrated in Figure 6. As introduced previously, we have 9 sub-matrix, each represents the corresponding decomposition in one direction. A_1 shows the sub-matrix of the upper-left direction. In addition, the network architecture of Auto-Encoder used in the reconstruction experiments are visualized in Figure 7.

The evaluation results of GCN, CNN and GraphCNN are summarized in Figure 2, 3, 4 and 5 for **Original**, **Stride**, **Stride+Skip** and **Stride+Skip+AP** settings respectively. Moreover, we evaluate the performances of randomly decomposed GraphCNNs and compare them with GCN and the GraphCNN decomposed by human prior. All the experiments are conducted on the network architectures of 17-layers in different settings (see Table 1).

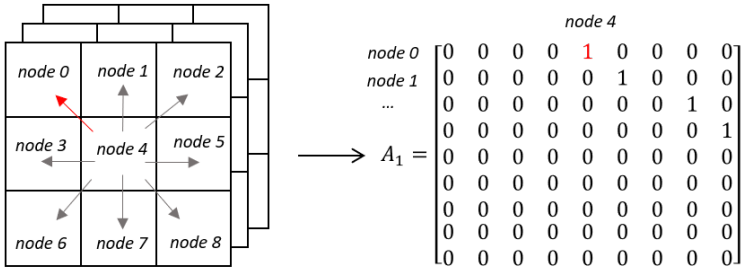


Figure 6: The left is a image with $3 \times 3 \times 3$ shape. 9 arrows in the image represent 9 edge type between nodes. A_1 matrix in the right represents the adjacency matrix composed of the direction with red arrow.

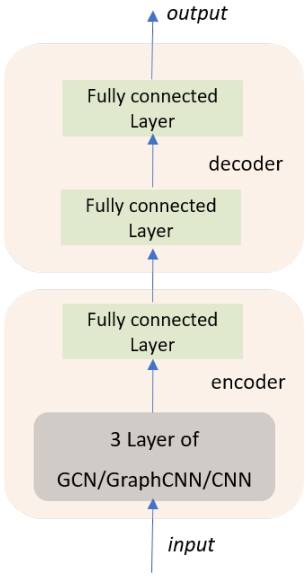


Figure 7: The network architecture of Auto-Encoder

Models	**GCN** (*Train*, *Test Acc*)	**GraphCNN** (*Train*, *Test Acc*)	**CNN** (*Train*, *Test Acc*)	**LSGCN** (*Train*, *Test Acc*)
1 Layer	46.4%, 48.8%	69.4%, 66.1 %	66.7%, 68.3%	46.1%, 49.5%
2 Layer	47.1%, 49.8%	81.1%, 80.6%	81.7%, 82.7%	49.1%, 52.3%
5 Layer	56.9%, 57.0%	97.2%, 89,9%	93.7%, 86.6%	61.0%, 58.6%
9 Layer	56.7%, 57.1%	99.0%, 90.3%	99.7%, 89.7%	70.2%, 59.8%
13 Layer	56.8%, 56.9%	99.6%, 90.0%	99.04%, 87.8%	77.0%, 58.9%
17 Layer	56.8%, 56.2%	99.1%, 88.6%	99.6%, 88.6%	79.1%, 59.3%

Table 2: Comparisons of different models on various depths (**Original Setting**)

Models	**GCN** (*Train*, *Test Acc*)	**GraphCNN** (*Train*, *Test Acc*)	**CNN** (*Train*, *Test Acc*)	**LSGCN** (*Train*, *Test Acc*)
5 Layer	54.9%, 56.0%	97.3%, 89.0%	99.4%, 89.0%	61.2%, 59.5%
9 Layer	63.8%, 60.2%	99.8%, 91.9%	99.6%, 91.4%	78.1%, 61.4%
13 Layer	82.1%, 57.8%	99.9%, 92.9%	100%, 93.1%	96.4%, 59.0%
17 Layer	88.9%, 53.1%	99.9%, 93.1%	100%, 93.1%	98.5%, 54.0%

Table 3: Comparisons of different models on various depths (**Stride Setting**)

Models	**GCN** (*Train*, *Test Acc*)	**GraphCNN** (*Train*, *Test Acc*)	**CNN** (*Train*, *Test Acc*)	**LSGCN** (*Train*, *Test Acc*)
5 Layer	59.1%, 58.8%	99.8%, 89.9%	99.5%, 88.7%	67.6%, 63.9%
9 Layer	65.8%, 63.0%	100%, 93.2%	99.9%, 91.0%	84.5%, 67.1%
13 Layer	73.5%, 64.4%	100%, 94.3%	100%, 93.0%	98.0%, 67.3%
17 Layer	80.0%, 63.5%	100%, 94.5%	100%, 93.2%	99.5%, 65.6%

Table 4: Comparisons of different models on various depths (**Stride+Skip Setting**)

Models	**GCN** (*Train*, *Test Acc*)	**GraphCNN** (*Train*, *Test Acc*)	**CNN** (*Train*, *Test Acc*)	**LSGCN** (*Train*, *Test Acc*)
5 Layer	64.7%, 63.7%	97.2%, 89.2%	99.5%, 88.7%	67.4%, 65.5%
9 Layer	81.4%, 72.0%	99.8%, 92.1%	99.9%, 91.9%	89.1%, 73.8%
13 Layer	90.5%, 74.7%	99.9%, 93.1%	100%, 92.9%	98.5%, 78.6%
17 Layer	94.0%, 72.8%	99.9%, 93.2%	100%, 93.2%	97.2%, 75.4%

Table 5: Comparisons of different models on various depths (**Stride+Skip+AP Setting**)

GCN	**GraphCNN**	**CNN**
0.818	0.781	0.774

Table 6: Reconstruction error of different models