DEEP ATTENTION NETWORKS FOR IMAGES AND GRAPHS

A Dissertation

by

ZHENGYANG WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2020

Major Subject: Computer Science

ABSTRACT

Deep learning has achieved great success in various machine learning areas, such as computer vision, natural language processing, and graph representation learning. While numerous deep neural networks (DNNs) have been proposed, the set of fundamental building blocks of DNNs remains small, including fully-connected layers, convolutions and recurrent units. Recently, the attention mechanism has shown promise in serving as a new kind of fundamental building blocks. Deep attention networks (DANs), *i.e.* DNNs that use the attention mechanism as a fundamental building block, have revolutionized the area of natural language processing. However, developing DANs for computer vision and graph representation learning applications is still challenging. Due to the intrinsic differences in data and applications, directly migrating DANs from textual data to images and graphs is usually either infeasible or ineffective. In this dissertation, we address this challenge by analyzing the functionality of the attention mechanism and exploring scenarios where DANs can push the limits of current DNNs. We propose several effective DANs for images and graphs.

For images, we build DANs for a variety of image-to-image transformation applications by proposing powerful attention-based building blocks. First, we start the exploration through studying a common problem in dilated convolutions, which naturally results in the use of the attention mechanism. Dilated convolutions, a variant of convolutions, have been widely applied in deep convolutional neural networks (DCNNs) for image segmentation. However, dilated convolutions suffer from the gridding artifacts, which hampers the performance. We propose two simple yet effective degridding methods by studying a decomposition of dilated convolutions, and generalize them by defining separable and shared (SS) operators. Then we connect the SS operators with the attention mechanism and propose the SS output layer, which is able to smooth the entire DCNNs by only replacing the output layer and improves the performance significantly. Second, we notice an interesting fact from the first study that, as the attention mechanism allows the SS output layer to have a receptive field of any size, the best performance is achieved when using a global receptive field. This fact motivates us to think of the attention mechanism as global operators, as opposed to local operators like convolutions. With this insight, we propose the non-local U-Nets, which are equipped with flexible attention-based global aggregation blocks, for biomedical image segmentation. In particular, we are the first to enable the attention mechanism for down-sampling and up-sampling processes. Finally, we go beyond biomedical image segmentation and extend the non-local U-Nets to global voxel transformer networks (GVTNets), which serve as a powerful open-source tool for 3D image-to-image transformation tasks. In addition to leveraging the non-local property of the attention mechanism under the supervised learning setting, we also investigate the generalization ability of the attention mechanism under the transfer learning setting. We perform thorough experiments on a wide range of real-world image-to-image transformation tasks, whose results clearly demonstrate the effectiveness and efficiency of our proposed DANs.

For graphs, we develop DANs for both graph and node classification applications. First, we focus on graph pooling, which is necessary for graph neural networks (GNNs) to perform graph classification tasks. In particular, we point out that the second-order pooling naturally satisfies the requirement of graph pooling but encounters practical problems. To overcome these problems, we propose attentional second-order pooling. Specifically, we bridge the second-order pooling with the attention mechanism and design an attention-based pooling method that can be flexibly used as either global or hierarchical graph pooling. Second, on node classification tasks, we pay attention to the problem that most GNNs lack the ability of performing effective non-local aggregation, which greatly limits the performance on disassortative graphs. In particular, it even leads to worse performance of GNNs than simple multi-layer perceptrons on some disassortative graphs. In order to address this problem, we propose a simple yet effective non-local aggregation framework with an efficient attention-guided sorting for GNNs, based on which we develop non-local GNNs. Experimental results on various graph and node classification benchmark datasets show that our DANs improve the performance significantly and consistently.

DEDICATION

To my wife, Min Shi,

my daughter, Coco Min Wang,

and my parents, Hong Zhu and Zhongguang Wang.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

I would like to express my sincere thanks to my advisor, Dr. Shuiwang Ji, for his guidance, support, and encouragement. During my research career, Dr. Ji has provided invaluable suggestions with his expertise and experience. His insights and passion are unparalleled. This Ph.D. journey with him will be a life-long asset of mine.

It is my great pleasure to have Dr. Nima Kalantari, Dr. Xia Hu, and Dr. Jianhua Huang on my dissertation committee. They have provided insightful suggestions and discussions on my research projects and dissertation.

I would like to thank Dr. Lei Cai, Dr. Hongyang Gao, Hao Yuan, Yi Liu, Yaochen Xie, Xinyi Xu, Meng Liu, Youzhi Luo, Limei Wang, and Zhao Xu, in the Data Integration, Visualization, and Exploration (DIVE) lab at TAMU, for their helpful discussions and collaborations. I would also like to thank other collaborators, Dr. Na Zou, Dr. Xia Hu, Dr. Shaoting Zhang, and Dr. Dinggang Shen.

Special thanks go to my friends, Yaqing Wang, Di Zhang, Can Zhang, Yuanji Xie, Yanni Ma, for their help and mental support.

Last but not least, I appreciate my family members, and in particular, my beloved wife, Min Shi. Without their continuous support and encouragement, I would not have finished my Ph.D. study.

Funding Sources

This work was supported in part by National Science Foundation grants IIS-1633359, IIS-1900990, IIS-1908166, IIS-1908198, IIS-1908220, DBI-1922969, National Institutes of Health grant 1R21NS102828, and Defense Advanced Research Projects Agency grant N66001-17-2-4031.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

The advances of deep learning [7] have revolutionized the area of machine learning and artificial intelligence. Deep neural networks (DNNs) have set new performance records in various fields, such as computer vision [8, 9, 10], natural language processing [11, 12, 13, 14], and graph representation learning [15, 16, 17, 18]. The most popular DNNs are convolutional neural network (CNNs) [8] and recurrent neural networks (RNNs) [11], which are mainly composed of a small set of fundamental building blocks, including fully-connected layers, convolutions and recurrent units. While the combination of these fundamental building blocks under different network architectures has resulted in numerous powerful DNNs, a new fundamental building block may enable more possibilities and greatly push the limits of DNNs. In recent years, the attention mechanism [19, 20, 13] has shown promise in serving as a new kind of fundamental building blocks.

The attention mechanism was initially proposed as an auxiliary block in encode-decoder networks [20, 21] or deep learning models with multiple input sources [19, 22, 23, 24, 25]. Generally, the attention mechanism allows feature vectors to interact with each other and uses the interaction results to guide feature aggregation and updating. DNNs with the attention mechanism have achieved great success in applications like machine translation [20, 13], speech recognition [21], text classification [26], language modeling [14], image captioning [19, 24], and visual question answering [22, 23, 25]. Recently, a fully attention-based network, called the Transformer [13], has been developed, where no recurrence or convolution exists. The Transformer outperforms all previous models and becomes the backbone model in many natural language processing applications [14].

We name DNNs that use the attention mechanism as a fundamental building block as deep attention networks (DANs). The success of the Transformer on textual data has motivated the research community to develop DANs for other fields as well, like computer vision and graph representation learning. However, extending the success of the Transformer to new fields is challenging due to the intrinsic differences in data and applications. For example, directly migrating

DANs from textual data to images and graphs is usually either infeasible or ineffective.

1.1 Dissertation Outline

In this dissertation, we explore efficient and effective DANs for image and graph tasks [27, 28, 29, 30, 31]. In particular, we aim at finding intuitive and effective way to apply the attention mechanism as a fundamental building block through analyzing its functionality. Specifically, Chapters 2, 3, and 4 introduce our DANs for images, while our DANs for graphs are covered by Chapters 5 and 6.

In Chapter 2, we start our exploration by solving the gridding problem in dilated convolutions. Through the process of developing degridding solutions, we introduce the use of the attention mechanism naturally and intuitively. First, we study a decomposition of dilated convolutions and propose two simple yet effective degridding methods, namely group interaction layers and separable and shared (SS) convolutions. Unlike existing models, which explore degridding solutions by focusing on a block of cascaded dilated convolutional layers, our methods address the gridding artifacts by smoothing the dilated convolution itself. We evaluate our proposed methods thoroughly, and visualize the smoothing effect through effective receptive field analysis. The experimental results demonstrate the superiority of our degridding methods. Second, we perform detailed analysis on the proposed methods and point out that the proposed two degridding approaches are intrinsically related. Based on this insight, we define general SS operators, which generalize the proposed methods. Next, we further explore SS operators in view of operators on graphs and propose the SS output layer, which employs the attention mechanism. The SS output layer is able to smooth the entire DCNNs with dilated convolutions by only replacing the output layer. Experimental results indicate that the SS output layer improves the performance significantly and has much fewer training parameters than the original output layer. Finally, as the attention mechanism allows the SS output layer to have a receptive field of any size, we explore how the size of the receptive field affects the performance. We find out that a global receptive field yields the best performance. This fact motivates the next chapter, where the attention mechanism is used as powerful global operators.

In Chapter 3, we take advantage of the property of the attention mechanism that it can be treated as global operators, and develop powerful non-local U-Nets for biomedical image segmentation tasks. Global operators refer to operators that have a global receptive field. For example, fully-connected layers are global operators. However, fully-connected layers have an excessive number of training parameters and are prone to over-fitting due to lack of regularization. On the contrary, the attention mechanism is light-weight and effective. To build our non-local U-Nets for biomedical image segmentation tasks, we propose flexible attention-based global aggregation blocks, which can be inserted into U-Net as size-preserving layers, as well as down-sampling and up-sampling layers. We are the first to extend the attention mechanism for down-sampling and up-sampling processes. Thorough experiments are performed on the 3D multimodality isointense infant brain MR image segmentation task, in order to evaluate our non-local U-Nets. Our proposed non-local U-Nets achieve top performances with better efficiency in terms of parameters and computation time. The results demonstrate the power of our flexible attention-based global aggregation blocks.

In Chapter 4, we extend the success of our non-local U-Nets to other 3D image-to-image transformation applications. In particular, we design more effective and efficient attention-based blocks named global voxel transformer operators (GVTOs) and develop an open-source tool named global voxel transformer networks (GVTNets). On several real-world augmented microscopy tasks, GVTNets show the ability of obtaining high-quality microscope images without using expensive microscopy hardware and sample preparation techniques, with improved performance over existing tools. In addition, we also investigate the generalization ability of the attention mechanism under the transfer learning setting, leveraging the fact that the attention mechanism processes inputs with input-dependent weights. Experimental results on three tasks under the transfer learning setting indicate clear performance boosts.

In Chapter 5, we focus on graph pooling for graph neural networks (GNNs) and connect the second-order pooling with the attention mechanism, which leads to the proposal of our attentional second-order pooling. GNNs have achieved great success in learning node representations for graph tasks such as node classification and link prediction. Graph classification requires graph pooling to obtain graph representations from node representations. It is challenging to develop graph pooling methods due to the variable sizes and isomorphic structures of graphs. In this work, we propose to use second-order pooling as graph pooling, which naturally solves the above challenges. In addition, compared to existing graph pooling methods, second-order pooling is able to use information from all nodes and collect second-order statistics, making it more powerful. We show that direct use of second-order pooling with GNNs leads to practical problems. To overcome these problems, we propose two novel global graph pooling methods based on second-order pooling; namely, bilinear mapping and attentional second-order pooling. In addition, we extend attentional second-order pooling to hierarchical graph pooling for more flexible use in GNNs. We perform thorough experiments on graph classification tasks to demonstrate the effectiveness and superiority of our proposed methods. Experimental results show that our methods improve the performance significantly and consistently.

In Chapter 6, we pay attention to applying GNNs on disassortative graphs and develop non-local GNNs. Modern GNNs learn node embeddings through multi-layer local aggregation and achieve great success in applications on assortative graphs. However, tasks on disassortative graphs usually require non-local aggregation. In addition, we find that local aggregation is even harmful for some disassortative graphs. In this work, we propose a simple yet effective non-local aggregation framework with an efficient attention-guided sorting for GNNs. Based on it, we develop various non-local GNNs. We perform thorough experiments to analyze disassortative graph datasets and evaluate our non-local GNNs. Experimental results demonstrate that our non-local GNNs significantly outperform previous state-of-the-art methods on six benchmark datasets of disassortative graphs, in terms of both model performance and efficiency.

1.2 Contributions

The main contributions of this dissertation can be summarized as below:

- We propose the separable and shared (SS) output layer based on the attention mechanism, through our our exploration of solutions to the gridding problem in dilated convolutions. The SS output layer is able to smooth the entire DCNNs with dilated convolutions by only replacing the output layer, yielding significantly improved performance.

- We propose the non-local U-Nets for biomedical image segmentation. The non-local U-Nets are built upon flexible attention-based global aggregation blocks, which can be used as size-preserving layers, as well as down-sampling and up-sampling layers. We are the first to extend the attention mechanism for down-sampling and up-sampling processes. Experimental results show that our proposed non-local U-Nets are able to achieve top performance with fewer parameters and faster computation than the original U-Net.
- We propose the global voxel transformer networks (GVTNets) for augmented microscopy. GVTNets extend the success of our non-local U-Nets to a wider range of 3D image-to-image transformation applications, with more effective and efficient attention-based global voxel transformer operators (GVTOs). We apply GVTNets on existing datasets for three different augmented microscopy tasks under both supervised and transfer learning settings. The performance is significantly and consistently better than previous U-Net based approaches.
- We propose the attentional second-order pooling for graph neural networks (GNNs), which can be flexibly used as either global or hierarchical graph pooling. Second-order pooling naturally solves the challenges of graph pooling and is more powerful with its ability of using information from all nodes and collecting second-order statistics. We address the practical problems in directly using second-order pooling and connect the second-order pooling with the attention mechanism, which leads to the proposal of our attentional second-order pooling.
- We propose the non-local GNNs for disassortative graphs. Modern GNNs learn node embeddings through multi-layer local aggregation and lack the ability of performing non-local aggregation, which is essential for tasks on disassortative graphs. To address this problem, we propose a simple yet effective non-local aggregation framework with an efficient attention-guided sorting for GNNs. Based on it, we develop various non-local GNNs.

2. DEEP ATTENTION NETWORKS FOR IMAGES: SMOOTHED DILATED CONVOLUTIONS FOR IMPROVED DENSE PREDICTION

In this chapter, we start with addressing the gridding problem in dilated convolutions, and then find a natural and intuitive way to use the attention mechanism in deep convolutional neural networks with dilated convolutions for dense prediction tasks. In particular, we propose the separable and shared (SS) output layer based on the attention mechanism, which is able to smooth the entire DCNNs with dilated convolutions by only replacing the output layer and yields significantly improved performance.*

2.1 Introduction

Dilated convolutions, also known as atrous convolutions, have been widely explored in deep convolutional neural networks (DCNNs) for various tasks, including semantic image segmentation [32, 33, 34, 35, 36, 37, 38, 39, 40, 41], object detection [42, 43, 44, 45], audio generation [46], video modeling [47], and machine translation [48]. The idea of dilated filters was developed in the *algorithm à trous* for efficient wavelet decomposition in [49] and has been used in image pixel-wise prediction tasks to allow efficient computation [32, 33, 42, 43]. Dilation upsamples convolutional filters by inserting zeros between weights, as illustrated in Figure 2.1. It enlarges the receptive field, or field of view [36, 37, 39], but does not require training extra parameters in DCNNs. Dilated convolutions can be used in cascade to build multi-layer networks [46, 47, 48]. Another advantage of dilated convolutions is that they do not reduce the spatial resolution of responses. This is a key difference from down-sampling layers, such as pooling layers or convolutions with stride larger than one, which expand the receptive field of subsequent layers as well but also reduce the spatial resolution. This allows the transfer of classification models trained on ImageNet [50, 10] to semantic image segmentation tasks by removing down-sampling layers and applying dilation

*Part of the data reported in this chapter is reprinted with permission from“Smoothed dilated convolutions for improved dense prediction." by Zhengyang Wang and Shuiwang Ji, 2018, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, vol. 1, pp. 2486-2495, Copyright 2018 by ACM.

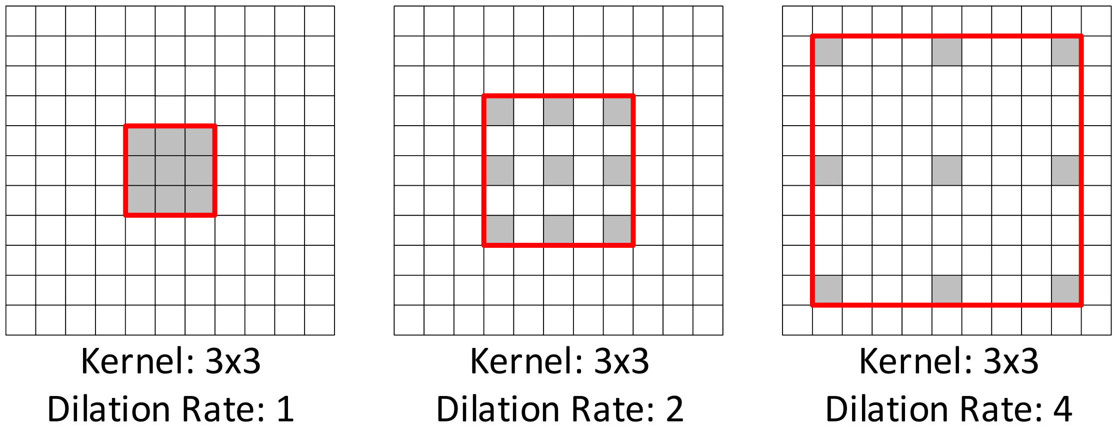


Figure 2.1: An illustration of 2-D Dilated convolutions with a kernel size of 3×3. Note that when the dilation rate is 1, dilated convolutions are the same as standard convolutions. Dilated convolutions enlarge the receptive field while keeping the spatial resolution.

in convolutions of subsequent layers [51, 34, 35, 36, 37, 38, 39, 40]. Similar to standard convolutions, a layer consisting of a dilated convolution with an activation function is called a dilated convolutional layer.

While DCNNs with dilated convolutions achieved success in a wide variety of deep learning tasks, it has been observed that dilations result in the so-called "gridding artifacts" [35, 38, 39]. For dilated convolutions with dilation rates larger than one, adjacent units in the output are computed from completely separate sets of units in the input. It results in inconsistency of local information and hampers the performance of DCNNs with dilated convolutions. As dilated convolutional layers are commonly stacked together in cascade in DCNNs, existing models focus on smoothing such gridding artifacts for a block of cascaded dilated convolutional layers. In [35, 39] the gridding problem was alleviated by adding more layers with millions of extra training parameters after the block of dilated convolutions. In [38] the hybrid dilated convolution (HDC) was proposed, which applies different dilation rates without a common factor for continuous dilated convolutional layers.

In this chapter, we address the gridding artifacts by smoothing the dilated convolution itself, instead of a block of stacked dilated convolutional layers. Our methods enjoy the unique advantage of being able to replace any single dilated convolutional layer in existing networks as they do not rely on other layers to solve the gridding problem. More importantly, our methods add minimal numbers of extra parameters to the model while some other degridding approaches increase the model parameters dramatically [35, 39]. Our methods are based on an interesting view of the dilated convolutional operation [52, 36, 53], which benefits from a decomposition of the operators. Based on this novel interpretation of dilated convolutions, we propose two simple yet effective methods to smooth the gridding artifacts. By analyzing these two methods in both the original operation and the decomposition views, we further notice that they are intrinsically related and define separable and shared (SS) operators that generalize the proposed methods. Experimental results show that our methods improve current DCNNs with dilated convolutions significantly and consistently, while only adding a few hundred extra parameters. We also employ the effective receptive field (ERF) analysis [54] to visualize the smoothing effect for DCNNs with our dilated convolutions.

Afterwards, we perform further analysis on SS operators in view of operators on graphs. Based on this analysis, we incorporate deep learning techniques on graphs and propose the SS output layer, which smooths DCNNs with dilated convolutions by only replacing the output layer. In addition, the SS output layer shows a better ability of aggregating information from large receptive fields than original output layers based on dilated convolutions. The smoothed DCNNs are able to produce significantly improved dense prediction.

2.2 Related Work

In this section, we describe dilated convolutions and DCNNs with them. We then discuss the gridding problem and current solutions in detail.

2.2.1 Dilated Convolutions

In the one-dimensional case, given a 1-D input f, the output o at location i of a dilated convolution with a filter w of size S is defined as

$$o[i] = \sum_{s=1}^{S} f[i + r \cdot s]w[i], \tag{2.1}$$

where r is known as the dilation rate. Higher dimensional cases can be easily generalized. When $r = 1$, dilated convolutions correspond to standard convolutions. An intuitive and direct way to understand dilated convolutions is that $r - 1$ zeros are inserted between every two adjacent weights in the standard convolutional filters. Dilated convolutions are also known as atrous convolutions in which "trous" means holes in French. Figure 2.1 contains an illustration of dilated convolution in the two-dimensional case.

As mentioned in Section 2.1, in most cases, DCNNs use dilated convolutions in cascade, which means several dilated convolutional layers are stacked together. The reasons for using this cascaded pattern differ for different tasks. In the task of semantic image segmentation [51, 34, 35, 36, 37, 38, 39, 40], in order to have output feature maps of larger sizes while maintaining the size of the receptive field, dilated convolutions are employed to replace standard convolutions in layers after the removed down-sampling layers. For example, if we treat standard convolutions as dilated convolutions with a dilation rate of $r = 1$, when a down-sampling layer with a subsampling rate of 2 is removed, the dilation rates of all subsequent convolutional layers should be multiplied by 2. This results in dilated convolutional layers with dilation rates of $r = 2, 4, 8$, etc. In other tasks, such as audio generation [46], video modeling [47], and machine translation [48], the use of dilated convolutions aims at enlarging the receptive fields of outputs. As pointed out in [34, 46, 47], cascaded dilated convolutional layers expand the receptive field exponentially in the number of layers in DCNNs, as opposed to linearly. In these studies, the dilation rate is doubled for every forward layer, starting from 1 up to a limit before the pattern is repeated.

Note that when using dilated convolutions in cascade, the gridding artifacts affect the models

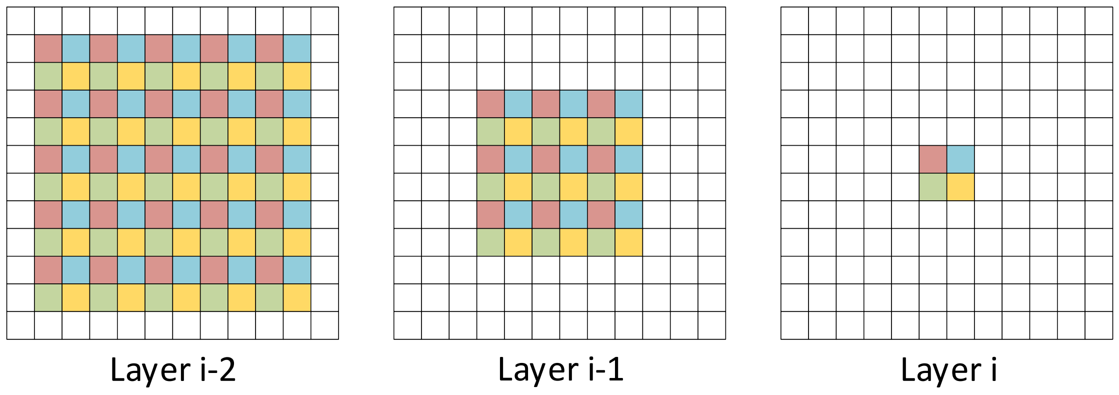


Figure 2.2: An illustration of gridding artifacts. The operators between layers are both dilated convolutions with a kernel size of 3×3 and a dilation rate of $r = 2$. For four neighboring units in layer i indicated by different colors, we mark their actual receptive fields in layer $i - 1$ and $i - 2$ using the same color, respectively. Clearly, their actual receptive fields are completely separate sets of units.

more significantly. This is because the dilation rates of continuously stacked layers have a common factor of 2 in all of these DCNNs that use dilated convolutional layers in cascade, as discussed in [38] and Section 2.2.2. In [36, 37] dilated convolutions in parallel to form the output layer were explored.

2.2.2 Gridding in Dilated Convolutions

Dilated convolutions with dilation rates larger than one will produce the so called gridding artifacts; that is, adjacent units in the output are computed from completely separate sets of units in the input and thus have totally different actual receptive fields. To view the gridding problem clearly, we first look into a single dilated convolution. Considering the second case in Figure 2.1 as an example, a 2-D dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 2$ has a 5×5 receptive field. However, the number of pixels that are actually involved in the computation is only 9 out of 25, which implies that the actual receptive field is still 3×3, but sparsely distributed. If we further consider the neighboring units in the output, the gridding problem can be seen from Figure 2.2. Suppose we have two consecutive dilated convolutional layers in cascade, and both dilated convolutions have a kernel size of 3×3 and a dilation rate of $r = 2$. For four adjacent units indicated by different colors in layer i, we show their actual receptive fields in layer $i-1$ and $i-2$ using the same color. We can see that four completely separate sets of units in layer $i-1$ contribute to the computation of the four units in layer i. Moreover, since the dilation rates for both layers are 2, which have a common factor of 2, the gridding problem also exists in layer $i-2$. Indeed, whenever the dilation rates of dilated convolutional layers in cascade have a common factor relationship, such as $2, 2, 2$ or $2, 4, 8$, the gridding problem is propagated to all layers, as pointed out in [38]. For a block of such layers, neighboring outputs of the block are computed from totally different sets of inputs. This results in the inconsistency of local information and hampers the performance of DCNNs with dilated convolutions.

The gridding artifacts were observed and addressed in several recent studies for semantic image segmentation [35, 38, 39]. As described in Section 2.2.1, dilated convolutions are mostly employed in cascade in DCNNs. Therefore, these studies focused on solving the gridding problem in terms of a block of stacked dilated convolutional layers. Specifically, hybrid dilated convolution (HDC) was proposed in [38], which groups several dilated convolutional layers and applies dilation rates without a common factor relationship. For example, for a block of dilated convolutions with a dilation rate of $r = 2$, every three consecutive layers are grouped together and the corresponding dilation rates are changed to $1, 2, 3$ instead of $2, 2, 2$. For a similar block with a dilation rate of $r = 4$, the same grouping principle is applied and the dilation rates become $3, 4, 5$, instead of $4, 4, 4$. When used together with their proposed dense up-sampling convolution (DUC), this approach improved DCNNs for semantic image segmentation. This strategy was also adopted as the "multigrid" method in recent work [37]. Prior to [38], the degridding was performed mainly by adding more layers after the block of dilated convolutional layers [35, 39]. It was proposed in [35] to add two more standard convolutional layers without residual connections while [39] proposed to add a block of dilated convolutional layers with decreasing dilation rates. The main drawback of such methods is the requirement for learning a large amount of extra parameters.

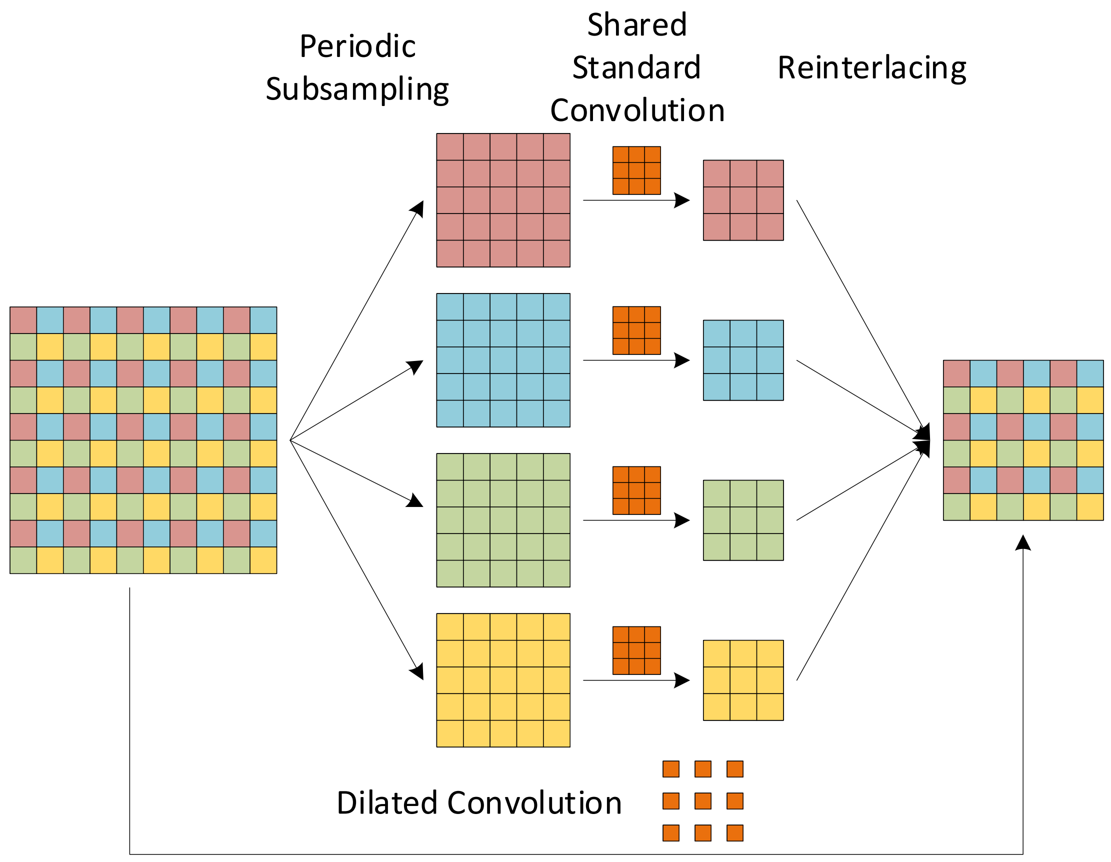


Figure 2.3: An example of the decomposition of a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 2$ on a 2-D input feature map. The decomposition has three steps; namely periodic subsampling, shared standard convolution and reinterlacing. This example will also be used in Figures 2.3 to 2.7.

2.3 Smoothed Dilated Convolutions

In this section, we discuss a decomposition view of dilated convolutions. We then propose two approaches for smoothing the gridding artifacts. We also analyze the relationship between the proposed two methods and define separable and shared (SS) operators to generalize them. Based on this analysis, we further propose the SS output layer to perform degridding for the entire network.

2.3.1 A Decomposition View of Dilated Convolutions

There are two ways to understand dilated convolutions. As introduced in Section 2.2.1, the first and more intuitive way is to think of dilated convolutional filters with dilation rate r as upsampled standard convolutional filters, by inserting zeros (holes) [43]. Another way to view dilated convolutions is based on a decomposition of the operation [52]. A dilated convolution with a dilation rate of r can be decomposed into three steps. First, the input feature maps are periodically subsampled by a factor of r. As a result, the inputs are deinterlaced to r^d groups of feature maps of reduced resolution, where d is the spatial dimension of the inputs. Second, these groups of intermediate feature maps are fed into a standard convolution. This convolution has filters with the same weights as the original dilated convolution after removing all inserted zeros. More importantly, it is shared for all the groups, which means each group of reduced resolution maps goes through the same standard convolution. The third step is to reinterlace the r^d groups of feature maps to the original resolution and produce the outputs of the dilated convolution.

Figure 2.3 gives an example of the decomposition in the 2-D case. To simplify the discussion, we assume the number of input channels and output channels is both 1. Given a 10×10 feature map, a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 2$ will output a 6×6 feature map without any padding. In the decomposition of this dilated convolution, the input feature map is periodically subsampled into $2^2 = 4$ groups of 5×5 feature maps of reduced resolution. Then a shared standard convolution, which has the same weights as the dilated convolution without padding, is applied to these 4 groups of feature maps and obtains 4 groups of 3×3 feature maps. Finally, they are reinterlaced to the original resolution and produce exactly the same 6×6 output feature map as the original dilated convolution. This decomposition reduces dilated convolutions into standard convolutions and allows more efficient implementation [32, 42, 36, 53].

We notice that the decomposition view provides a clear explanation of the gridding artifacts; that is, the r^d groups of intermediate feature maps, either before or after the shared standard convolution, have no dependency among each other and thus collect potentially inconsistent local information. Based on this insight, we overcome gridding by adding dependencies among the r^d

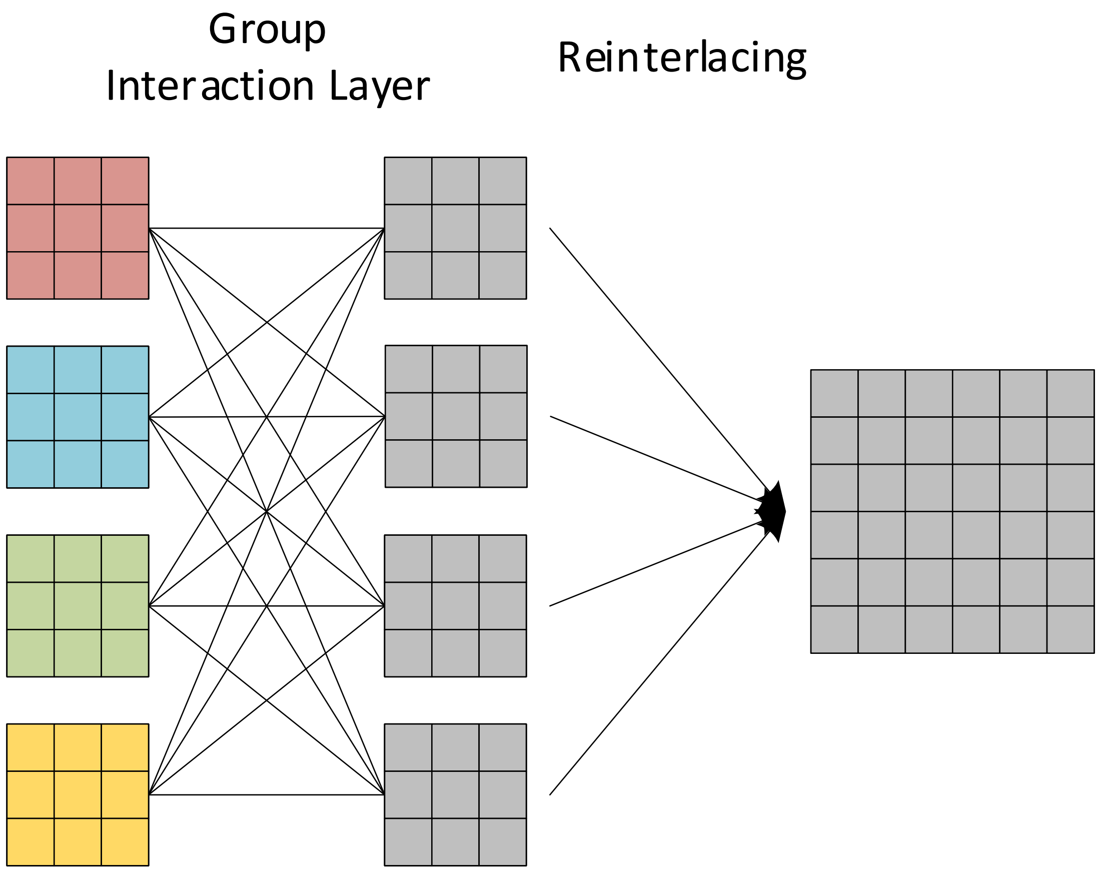


Figure 2.4: An illustration of the degridding method in Section 2.3.2 for a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 2$ on a 2-D input feature map. By using a group interaction layer before reinterlacing, dependencies among intermediate groups are established. The same gray color denotes consistent local information.

groups in different steps of the decomposition. We propose two effective approaches in the next two sections.

2.3.2 Smoothed Dilated Convolutions by Group Interaction Layers

Our first degridding method attempts to build dependencies among different groups in the third step of the decomposition. We propose to add a group interaction layer before reinterlacing the intermediate feature maps to the original resolution. For a dilated convolution with a dilation rate of r on d-dimensional input feature maps, the second step of the decomposition produces r^d groups of feature maps of reduced resolution, denoted as $\{f_i\}_{i=1}^{r^d}$, after the shared convolution. Note that each f_i represents a group of feature maps, rather than a single feature map. We define a group interaction layer with a weight matrix $W \in \mathbb{R}^{r^d \times r^d}$ given as

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1,r^d} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2,r^d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{r^d,1} & w_{r^d,2} & w_{r^d,3} & \cdots & w_{r^d,r^d} \end{bmatrix}. \tag{2.2}$$

The outputs of this layer are still r^d groups of feature maps, denoted as $\{\hat{f}_i\}_{i=1}^{r^d}$, computed by

$$\hat{f}_i = \sum_{j=1}^{r^d} w_{ij} \cdot f_j, \tag{2.3}$$

for $i = 1, 2, \ldots, r^d$. Note that the connections of this layer are between groups instead of feature maps. In fact, every $\hat{f}_i$ is a linear combination of $\{f_i\}_{i=1}^{r^d}$, weighted by the weight matrix W. Through this layer, each $\hat{f}_i$ collects local information from all r^d groups of feature maps, which adds dependencies among different groups. After the group interaction layer, the r^d groups are reinterlaced to the original resolution and form the final output of the dilated convolutions. The number of extra training parameters in such smoothed dilated convolutions is r^{2d}, independent of the number of input and output channels. DCNNs with dilated convolutions are commonly used in one-dimensional or two-dimensional cases, which means $d = 1, 2$. In practice, choices of r are usually $2, 4, 8$. The proposed group interaction layer only requires learning thousands of extra parameters in the worst cases, while the original dilated convolutions usually have millions of training parameters.

We use the same example in Section 2.3.1 to illustrate the idea in Figure 2.4. Given the outputs of the second step in the decomposition, the 4 groups of intermediate feature maps build dependencies among each other through the group interaction layer, whose number of weights is only $2^{2\cdot 2} = 16$, represented by 16 connections. We use the gray color to represent feature maps after degridding.

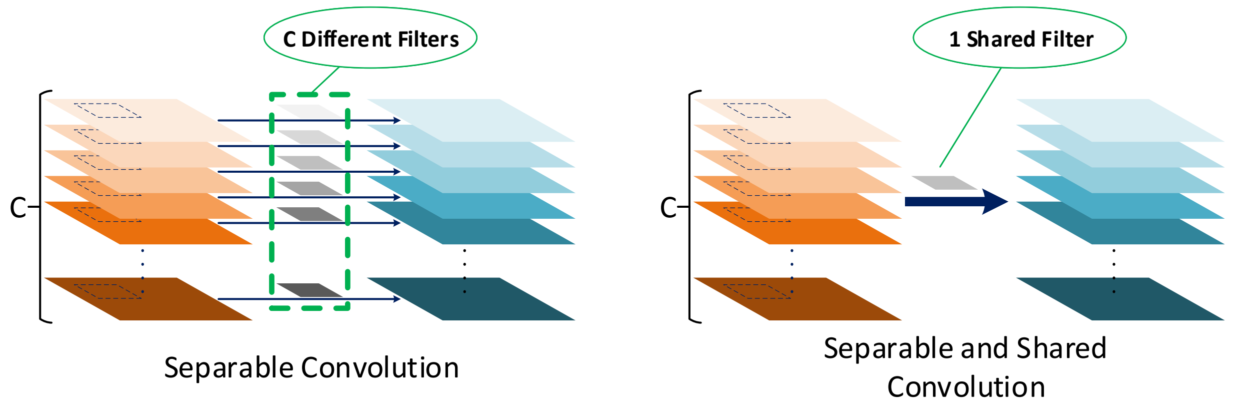


Figure 2.5: An illustration of the differences between the separable convolution and the proposed SS convolution introduced in Section 2.3.3. For inputs and outputs of C channels, the separable convolution has C filters in total, with one filter for each channel, while the SS convolution only has one filter shared for all channels.

2.3.3 Smoothed Dilated Convolutions by Separable and Shared Convolutions

We further explore an approach to establish dependencies among different groups in the first step of the decomposition; that is, before deinterlacing the input feature maps. Considering a dilated convolution with a dilation rate of r on d-dimensional input feature maps, the periodic subsampling during deinterlacing distributes each unit in a local area of size r^d in the inputs to a separate group. Therefore, for units in a particular group, all the neighboring units are in the other independent $r^d - 1$ groups, thereby resulting in local inconsistency. If the local information can be incorporated before periodic sampling, it is possible to alleviate the gridding artifacts.

In order to achieve this, we propose separable and shared (SS) convolutions, based on separable convolutions [55, 56]. Given inputs of C channels and corresponding outputs of C channels, separable convolutions are the same as standard convolutions, except that separable convolutions handle each channel separately. Standard convolutions connect all C channels in inputs to all C channels in outputs, leading to C^2 different filters. In contrast, separable convolutions only connect the ith output channel to the ith input channel, yielding only C filters. In the proposed SS convolutions, "shared" means that, based on separable convolutions, the C filters are the same and shared by all pairs of input and output channels. For inputs and outputs of C channels, SS convo-

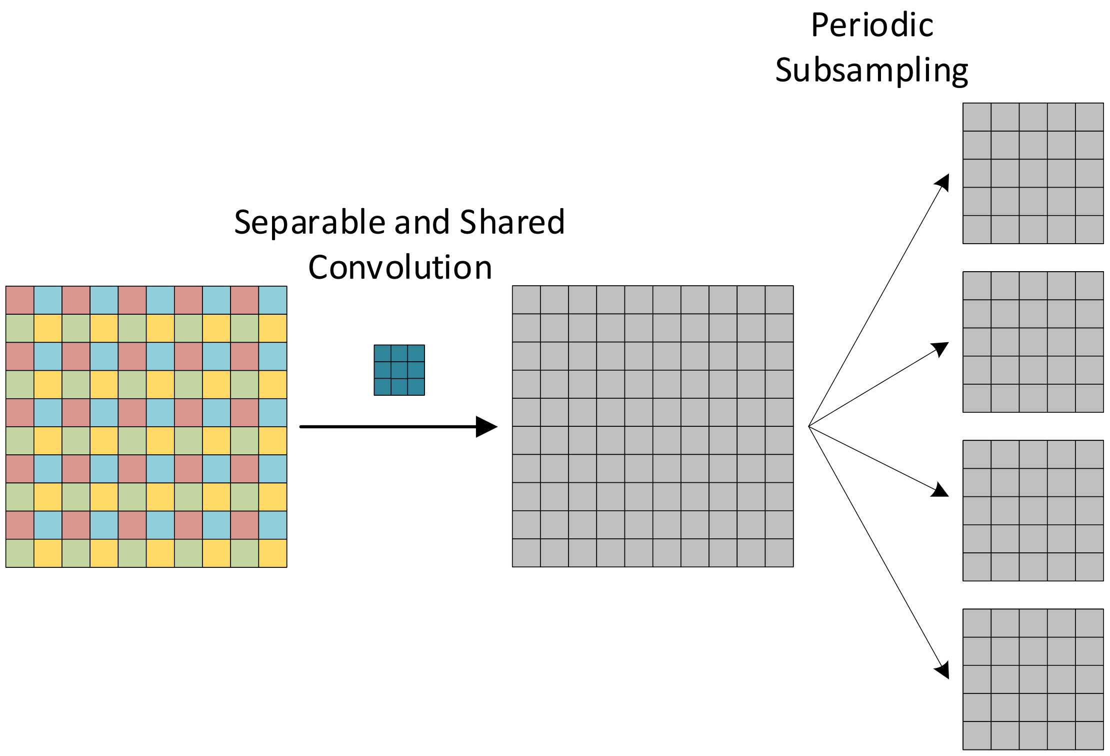


Figure 2.6: An illustration of the degridding method in Section 2.3.3 for a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 2$ on a 2-D input feature map. By adding the separable and shared convolution, the 4 groups created by periodic subsampling have dependencies among each other. The same gray color represents smoothed feature maps.

lutions only have one filter scanning all spatial locations and share this filter across all channels. Figure 2.5 provides a comparison between separable convolutions and SS convolutions. In terms of smoothing dilated convolutions, we apply SS convolutions to incorporate neighboring information for each unit in the input feature maps. Specifically, an SS convolution with a kernel size of $(2r-1)^d$ is inserted before deinterlacing, thereby adding dependencies among each other to the r^d groups of feature maps produced by periodic subsampling.

The example in Figure 2.6 illustrates the idea of inserting SS convolutions. Here, the kernel size of the inserted SS convolution is $(2 \cdot 2 - 1)^2 = 3 \times 3$. Note that because the inputs only have one channel, SS convolutions, separable convolutions and standard convolutions are equivalent in this example. However, they become different if the inputs have $C > 1$ channels. Importantly, for inputs with multiple channels, the number of training parameters does not change for SS convolutions, as opposed to the other two kinds of convolutions. It means the proposed degridding method has $(2r-1)^d$ parameters, independent of the number of channels, which corresponds to only tens of extra parameters at most in practice.

2.3.4 Relationship between the Two Methods

Both of the proposed approaches are derived from the decomposition view of dilated convolutions. Now we combine all steps and analyze them in view of the original operation. For the second method in Section 2.3.3, it is straightforward as the separable and shared (SS) convolution is inserted before the first step of decomposition and actually does not affect the original dilated convolution. Consequently, it is equivalent to adding an SS convolution before the dilated convolution, as shown in Figure 2.7. However, the first method in Section 2.3.2 performs degridding through the group-wise fully-connected layer between the second and the third steps of the decomposition. To see how to perform the combination, we refer to the example in Figure 2.4. Before the final step, we have four groups of feature maps and each group has only one feature map. Considering the units in the upper left corner of the four feature maps, without the group interaction layer, these four units form the upper left 2×2 block of the output feature map after reinterlacing. If we insert the group-wise fully-connected layer, the four new units in the upper left corner become linear combinations of the previous ones and form the upper left 2×2 block of the output feature map instead. As a result, the new upper left 2×2 block of the output feature map is computed by a fully-connected operation on the previous one. By examining other units, we find that the fully-connected operation is shared for every non-overlapping 2×2 blocks, scanning the output feature map with a stride of 2. Figure 2.8 provides an illustration. By generalizing this example, we can see that the degridding method is equivalent to a dilated convolution followed by the following operation: use a window of size r^d to scan the output feature map with stride r and obtain non-overlapping blocks; for each block, perform the same fully-connected operation that outputs a block of the same spatial size. Note that if the outputs have multiple channels, the operation is shared across channels. This operation is similar to the SS convolution as they both scan spatial

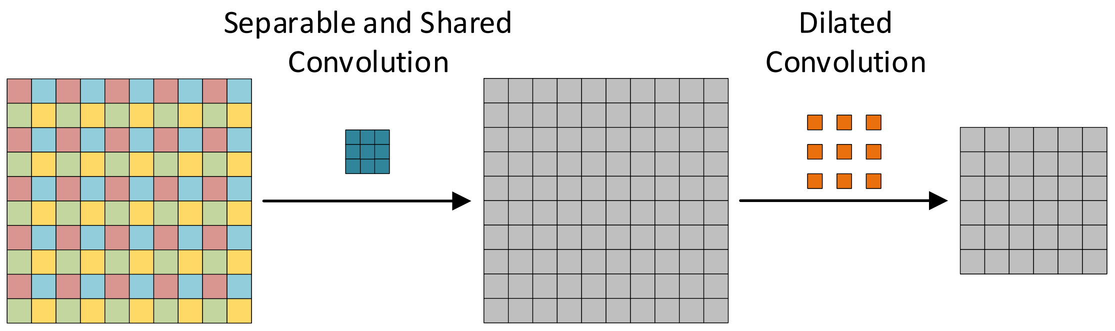


Figure 2.7: Another illustration of the proposed method in Section 2.3.3, corresponding to Figure 2.6. The method is equivalent to inserting an SS convolution before the dilated convolution.

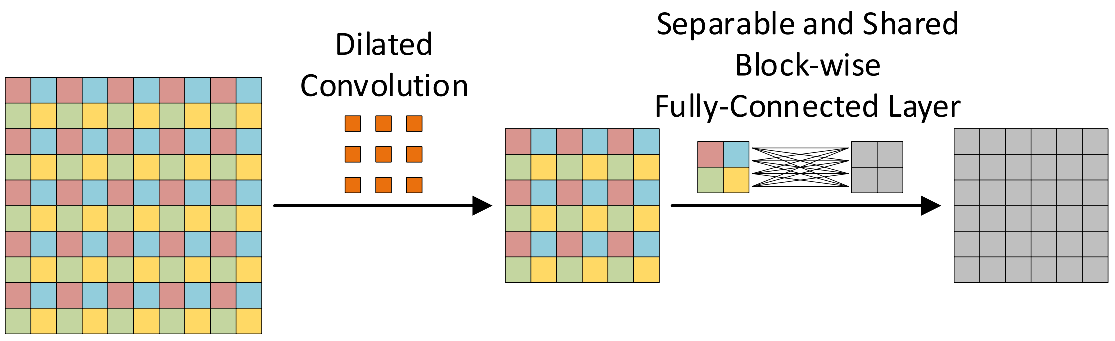


Figure 2.8: Another illustration of the proposed method in Section 2.3.2, corresponding to Figure 2.4. The method is equivalent to adding an SS block-wise fully-connected layer after the dilated convolution.

locations using a single kernel shared across all channels. Thus, we name it as the SS block-wise fully-connected layer. Based on it as well as the SS convolution, we further define operators which scan spatial locations of inputs using a single filter shared across all channels as SS operators.

As DCNNs commonly employ dilated convolutional layers in cascade, we also look into our proposed methods in this case. As explained above, the first degridding approach is equivalent to adding an SS block-wise fully-connected layer after the dilated convolution, while the second one corresponds to inserting an SS convolution before the dilated convolution. However, for a block of cascaded dilated convolutional layers with the same dilation rate, the order between the

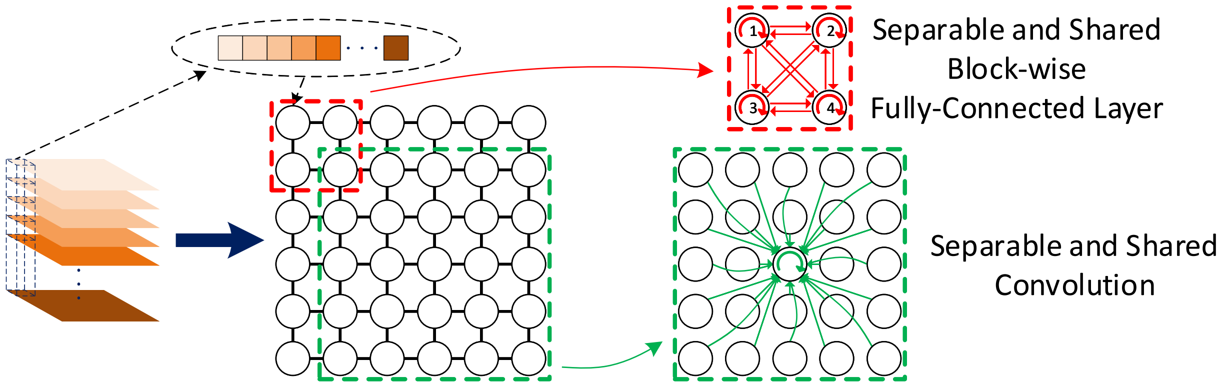


Figure 2.9: An illustration of SS operators in view of operators on graphs. Details are provided in Section 2.3.5. The circle arrow inside a node represents the self-loop.

dilated convolution and the SS operation only affects the very first and last layers. As a result, the two proposed degridding methods can be generalized as combining appropriate SS operators with dilated convolutions.

2.3.5 Separable and Shared Operators

With the insights above, we develop more effective SS operators to improve dense prediction models with dilated convolutions. According to the definition in Section 2.3.4, the key of SS operators is to apply a filter that is shared across all channels. Based on this property, we have reinvestigated SS operators in view of operators on graphs. Note that the data that we focus on in this chapter are grid-like data, such as 1-D text sequences, 2-D images, 3-D videos, etc. For inputs of C channels, each spatial location corresponds to a C-dimensional vector. By treating each vector as a node in a graph, the inputs are transformed into a grid-like graph. The left part of Figure 2.9 provides an illustration of this transformation for 2-D inputs. We first revisit the proposed SS block-wise fully-connected layer and SS convolution on this graph.

A 2×2 SS block-wise fully-connected layer scans the inputs using a 2×2 window with a stride of 2, as illustrated by the red box in Figure 2.9. To see the computation within the window, we denote the four nodes as n_1, n_2, n_3, n_4 as marked in the figure. The filter $W \in \mathbb{R}^{4\times 4}$ in this layer is given by

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix}. \tag{2.4}$$

The outputs $\hat{n}_1, \hat{n}_2, \hat{n}_3, \hat{n}_4$ of the window are computed by

$$\hat{n}_i = \sum_{j=1}^{4} w_{ij} \times n_j \tag{2.5}$$

for $i = 1, 2, \ldots, 4$. Here, $w_{ij} \times n_j$ means multiplying each element of n_j by w_{ij}, which is consistent with sharing W across all channels. In terms of operators on graphs, such computation can be interpreted as a process on a directed subgraph composed of the nodes in the scanning window. Every node interacts with each other and produces its new representation, where the interactions are modeled by directed edges. Specifically, the subgraph does not follow the original grid-like connections. Instead, each node has a directed edge to all nodes including itself, as shown by the top right part of Figure 2.9. Each directed edge represents a scalar weight in W. For example, the edge from n_3 to n_1 corresponds to w_{13}, measuring the importance of n_3 to $\hat{n}_1$. In other words, the SS block-wise fully-connected layer forms a fully-connected directed subgraph in each window during scanning.

The SS convolution differs from the SS block-wise fully-connected layer in that it constructs a different subgraph in its scanning window. The bottom right part of Figure 2.9 illustrates the directed subgraph for a 5×5 SS convolution. Unlike the SS block-wise fully-connected layer, where all nodes in a window get updated, the SS convolution only updates the representation of the center node by incorporating information from all nodes in the window. Therefore, the subgraph has directed edges from all nodes to the center node. Nodes except for the center node do not have self-loops or edges between each other. Again, each directed edge refers to a scalar weight. There are 25 directed edges corresponding to the 5×5 filter of the SS convolution.

To conclude, SS operators can be viewed as scanning the transformed graph using a window. Within each window, a directed subgraph is constructed, where each edge represents a scalar weight. Different ways to form the directed graph result in different SS operators. In addition, there are other ways to generate the scalar weights, instead of making them as training parameters. Many studies on deep learning on graphs have explored this direction, such as mixture model networks (MoNet) [57], GraphSAGE [16], graph attention networks (GAT) [17], and learnable graph convolutional networks (LGCN) [18]. In the next section, we incorporate deep learning techniques on graphs and propose an efficient and effective SS output layer, which improves DCNNs with dilated convolutions by simply replacing the output layer.

2.3.6 Smoothed DCNNs with Dilated Convolutions

The two proposed methods in Sections 2.3.2 and 2.3.3 are able to smooth any single dilated convolution. Our experimental results in Section 2.4 show that the proposed methods improve the encoders of DCNNs with dilated convolutions. However, dilated convolutions are also used in the output layer of these DCNNs. In this section, we explore the use of SS operators to smooth the entire network.

Various output layers have been proposed for DCNNs with dilated convolutions, in order to aggregate information from large receptive fields for prediction. For example, the large field of view (LargeFOV) layer in [36] is a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 12$ followed by 1×1 regular convolutions. The LargeFOV layer has been extended to the atrous spatial pyramid pooling (ASPP) layer [36, 37]. In the ASPP layer, four LargeFOV layers with different dilation rates are employed in parallel, and the outputs are summed or concatenated together as the final output. However, both output layers do not have any smoothing operation, thereby inheriting the gridding artifacts from the encoder to the final output, as illustrated by Figure 2.2 in Section 2.2.2.

To address this problem, we propose the SS output layer, which improves the performance by simply replacing dilated convolutions in the output layer by an appropriate SS operation. The proposed SS output layer is able to perform both smoothing and information aggregation for predic-

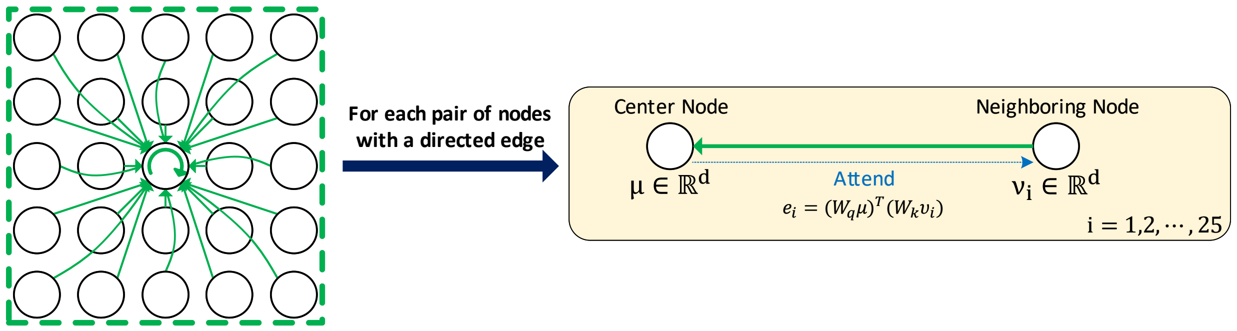


Figure 2.10: An illustration of our graph attention mechanism. Details are provided by Equations 2.6 to 2.8 in Section 2.3.6. Here, $s = 5$ so that $i = 1, 2, \dots, 25$.

tion. First, in Section 2.3.4, we conclude that the proposed degridding methods can be generalized as inserting SS operators between consecutive dilated convolutions. However, for DCNNs whose encoders use dilated convolutions in cascade, it may be more efficient to add only one SS operation after the entire encoder, making it a part of the output layer. Second, the analysis in Section 2.3.5 indicates that SS operators are able to aggregate information within each scanning window. The advantage of SS operators as compared with output layers based on dilated convolutions is that, given the same receptive field, information from all locations will be incorporated, instead of sampled ones. In addition, SS operators usually have much fewer parameters than dilated convolutions, as analyzed in Section 2.3.3. As a result, using the SS output layer is efficient and effective.

To be specific, we first transform the output feature maps of the encoder to the grid-like graph as shown in the left part of Figure 2.9. Then, we propose an SS operation that constructs the same subgraph within each window as the SS convolution, which is illustrated by the bottom right part of Figure 2.9. Differently, we adopt the graph attention mechanism in GAT [17] to generate the scalar weights. Suppose the window size of our SS operation is $s \times s$. There will be s^2 directed edges in the subgraph constructed by the scanning window. We denote the starting nodes of these directed edges as neighboring nodes $v_i, i = 1, 2, \dots, s^2$ and the center node as μ. Note that neighboring nodes include the center node. v_i and μ are d-dimensional vectors, where d is the number of input channels. For each directed edge, we compute attention coefficients defined as

$$e_i = (W_q\mu)^T(W_k v_i), \tag{2.6}$$

for $i = 1, 2, \ldots, s^2$. Here, $W_q, W_k \in \mathbb{R}^{d_k \times d}$ are shared for each edge and d_k is a hyperparameter. The attention coefficients are normalized across i, *i.e.,* all edges:

$$\alpha_i = Softmax(\frac{e_i}{\sqrt{d_k}}), \tag{2.7}$$

where α_i is the generated scalar weight corresponding to the i-th directed edge. The output of this window, which is the updated representation of the center node, is computed by

$$\hat{\mu} = \sum_{i=1}^{s^2} \alpha_i \times W_v v_i, \tag{2.8}$$

where $W_v \in \mathbb{R}^{d_o \times d}$ is also shared for each edge and d_o is a hyperparameter representing the dimension of $\hat{\mu}$. Figure 2.10 illustrates the graph attention process within a 5×5 scanning window. Note that if we choose the window size to be larger than the spatial sizes of inputs, the SS operation is able to aggregate global information for prediction. In addition, the SS operation has the same number of parameters when changing the window size, because W_q, W_k, W_v are all shared for edges, and different window sizes only result in different number of edges.

In our SS output layer, the proposed SS operation scans the transformed grid-like graph and updates every node. Appropriate padding is employed and α_i is always set to 0 for padding nodes. We also apply the multi-head attention as in GAT [17]. A 1×1 regular convolution follows the SS operation to produce the final output. The proposed SS output layer is evaluated in Section 2.4.5.

2.4 Experimental Studies

In this section, we evaluate our methods on the PASCAL VOC 2012 [58] and Cityscapes [59] datasets. Our proposed approaches result in significant and consistent improvements for DCNNs with dilated convolutions. We also perform the effective receptive field (ERF) analysis [54] to visualize the smoothing effect. Finally, we analyze the effectiveness and efficiency of the proposed separable and shared output layer.

2.4.1 Experimental Setup

To conduct our experiments, we choose the task of semantic image segmentation because the gridding artifacts were mainly observed in studies for this task [35, 38, 39]. The consistency of local information is important for such a pixel-wise prediction task on images. In addition, the smoothing effect is easy to visualize on two-dimensional data.

The baseline model in our experiments is the DeepLabv2 [36] with ResNet-101 [10]. It is a fair benchmark to evaluate our smoothed dilated convolutions in three aspects. First, it employed dilated convolutions to adapt ResNet pre-trained on ImageNet [50]; namely from image classification to semantic image segmentation. Most semantic image segmentation models adopted this transfer learning strategy [32, 33, 51, 34, 35, 36, 37, 38, 39, 40] and ResNet is one of the most accurate DCNNs for image classification with pre-trained models available. Second, models that achieved the state-of-the-arts in segmentation tasks recently [37, 38, 40] were developed from DeepLabv2. In [40] the output layer was replaced with a pyramid pooling module. [38] also changed the output layer and additionally proposed changing dilation rates, as mentioned in Section 2.2.2. The current best model [37] followed the suggestions of [38] and meanwhile, explored going deeper with more dilated convolutional blocks. Third, we intend to compare our degridding methods with existing approaches [35, 39, 38]. While [35, 39] addressed the gridding artifacts by adding more layers that considerably increased the number of training parameters, our methods only require learning hundreds of extra parameters. Thus, we perform the comparison with the idea proposed in [38], which is based on DeepLabv2.

DeepLabv2 is composed of two parts: the encoder and the output layers. The encoder is a pre-trained ResNet-101 model modified with dilated convolutions, and it extracts feature maps from raw images. As introduced in Section 2.2.1, the last two down-sampling layers in ResNet-101 were removed and subsequent standard convolutional layers were replaced by dilated convolutional layers with dilation rates of $r = 2, 4$, respectively. To be specific, after the modification, the last two blocks are a block of 23 stacked dilated convolutional layers with a dilation rate of $r = 2$ followed by a block of 3 cascaded dilated convolutions with a dilation rate of $r = 4$. The output layer performs pixel-wise classification by aggregating information from the output feature maps of encoder.

We re-implement DeepLabv2 in Tensorflow and perform experimental studies based on our implementation. Our code is publicly available[†]. We improve the baseline by addressing the gridding artifacts in the last two blocks of the encoder. To make the comparison independent of the output layer, we conduct experiments with different output layers. In order to eliminate the bias of different datasets, we evaluate our methods on two datasets. All the models are evaluated by pixel intersection over union (IoU) defined as

$$IoU = \frac{true\_positive}{true\_positive + false\_positive + false\_negative}. \tag{2.9}$$

2.4.2 PASCAL VOC2012

The PASCAL VOC 2012 semantic image segmentation dataset [58] provides pixel-wise annotated natural images. It has been split into *train*, *val* and *test* sets with $1,464$, $1,449$ and $1,456$ images, respectively. The annotations include 21 classes, which are 20 foreground object classes and 1 class for background. An augmented version with extra annotations [60] increases the size of the *train* set to $10,582$. In our experiments, we train all the models using the augmented *train* set and evaluate them on the *val* set. When reproducing the baseline DeepLabv2, we do not employ multi-scale inputs with max fusion for testing due to our limited GPU memory. We perform no post-processing such as conditional random fields (CRF) [36], which is not related to our goals. Following DeepLabv2, we train the model with randomly cropped patches of size of 321×321 and batch size of 10. Data augmentation by randomly scaling the inputs for training is applied. We

[†]`https://github.com/divelab/dilated/`

Table 2.1: Experimental results of models with the ASPP output layer and MS-COCO pre-training on PASCAL VOC 2012 *val* set. Class 1 is the background class and Class $2-21$ represent "aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, potteplant, sheep, sofa, train, tvmonitor", respectively. This is the same for Tables 2.1 to 2.7.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**mIoU**
DeepLabv2	93.8	85.9	38.8	84.8	64.3	**79.0**	93.7	85.5	**91.7**	34.1	83.0	57.0	**86.1**	83.0	81.0	85.0	58.2	83.4	**48.2**	**87.2**	74.0	75.1
Multigrid	93.6	85.4	38.9	82.2	66.9	76.6	93.2	85.3	90.7	35.7	82.5	53.7	83.1	**84.2**	82.2	84.6	56.9	84.3	45.6	85.5	73.1	74.5
G Interact	93.7	**86.9**	**39.6**	84.1	**68.9**	76.4	**93.8**	86.2	**91.7**	**36.1**	**83.7**	55.3	85.7	84.0	82.2	84.9	**59.5**	**85.7**	46.5	85.0	73.0	**75.4**
SS Conv	**93.9**	86.7	39.5	**86.2**	68.1	77.3	**93.8**	**86.4**	91.5	35.4	83.2	**59.0**	85.2	83.6	**82.4**	**85.2**	57.3	82.1	45.8	86.1	**75.2**	**75.4**

set the initial learning rate to 0.00025 and adopt the "poly" learning rate policy [61] as

$$current\_lr = (1 - \frac{iter}{max\_iter})^{power} \cdot initial\_lr, \tag{2.10}$$

where $power = 0.9$, $iter$ denotes current iteration number, and lr denotes learning rate, as in [36, 37, 38]. The model is trained for $max\_iter = 20,000$ iterations with a momentum of 0.9 and a weight decay of 0.0005.

We implement our proposed methods by inserting appropriate separable and shared (SS) operators before or after each dilated convolution as shown in Figures 2.8 and 2.7. An important step is to change the initial learning rate, detailed in each experiment. To make the comparisons solid, we also train the baseline with different initial learning rates and observe the original setting of 0.00025 yields the best performance. The initialization of SS operators is to set them to be identity

Table 2.2: Experimental results of models with the ASPP output layer but no MS-COCO pre-training on PASCAL VOC 2012 *val* set.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**mIoU**
DeepLabv2	92.9	85.0	38.1	82.8	66.2	76.5	91.1	82.7	88.4	**33.8**	77.7	49.9	80.7	78.6	77.9	82.0	51.5	76.6	43.1	82.8	66.6	71.7
Multigrid	92.8	84.9	37.4	81.8	65.6	76.0	90.4	81.3	86.9	32.6	76.8	52.3	80.2	79.5	77.4	81.9	50.7	78.4	41.9	82.7	66.0	71.3
G Interact	**93.0**	85.1	37.4	**83.4**	**66.9**	76.6	90.7	82.0	88.1	**33.8**	**81.1**	**54.3**	81.6	**80.2**	76.7	81.9	**53.7**	**78.7**	43.1	**83.9**	66.4	**72.3**
SS Conv	**93.0**	**85.8**	**38.3**	82.5	66.3	**77.9**	**91.6**	**83.5**	**88.5**	32.4	77.8	52.5	**81.9**	78.1	**79.3**	**82.1**	49.8	78.4	**44.4**	83.0	**67.9**	72.1

operators. Specifically, for a group interaction layer with a dilation rate of $r = 2$, the initial filter is

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.11}$$

while for an SS convolution with a dilation rate of $r = 2$, it is

$$W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{2.12}$$

The original DeepLabv2 used pre-training on MS-COCO [62], which results in more training data and higher performances. Our experiments are conducted under both settings; namely with and without MS-COCO pre-training. The results are given in Tables 2.1 and 2.2, respectively. In the tables, "G Interact" denotes the degridding method with a group interaction layer, *i*.e., adding

Table 2.3: Experimental results of models with the LargeFOV output layer and MS-COCO pre-training on PASCAL VOC 2012 *val* set.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**mIoU**
DeepLabv2	93.7	**85.7**	39.4	85.9	67.6	**79.0**	93.1	86.0	90.7	36.2	79.8	54.6	83.7	80.9	**81.4**	85.0	57.5	83.5	45.5	84.5	74.1	74.7
G Interact	**93.8**	85.5	**40.0**	86.5	67.5	78.1	92.9	86.2	90.4	**37.2**	80.6	**56.5**	82.6	80.3	81.0	85.0	58.1	**84.8**	**46.6**	84.4	74.8	74.9
SS Conv	**93.8**	85.3	39.7	**86.8**	**68.7**	77.9	**94.0**	**86.3**	**90.8**	35.2	**83.1**	55.4	**84.5**	**83.8**	79.6	**85.6**	**59.3**	83.2	46.2	**86.2**	**75.5**	**75.3**

an SS block-wise fully-connected layer after the dilated convolution and “SS Conv” represents the one with an SS convolution inserted before the dilated convolution. In these experiments with MS-COCO pre-training, the initial learning rates for “G Interact” and “SS Conv” are both 0.001. Otherwise, they are set to 0.001 and 0.00075, respectively. Clearly, both proposed methods improve the IoU for most classes as well as the mean IoU (mIoU) over the baseline under both settings. It is worth noting that “G Interact” only requires training $1,136(= 16 \times 23 + 256 \times 3)$ extra parameters and “SS Conv” requires $354(= 9 \times 23 + 49 \times 3)$ extra parameters, which are negligible compared to the total number of parameters in the models.

We also compare our methods with existing degridding method proposed in [38] and used in [37] as the “multigrid” method. As introduced in Section 2.2.2, the idea is to group several dilated convolutional layers and change the dilation factors. As we know, for the modified ResNet-101 with dilated convolutions, the last two blocks are a block of 23 stacked dilated convolutional layers with a dilation rate of $r = 2$ followed by a block of 3 cascaded dilated convolutions with a dilation rate of $r = 4$. For the first block, we group every 3 layers together and replace the dilation rates from $r = 2, 2, 2$ to $r = 1, 2, 3$. We keep $r = 2, 2$ for the left 2 layers. For the second block, the 3 dilation factors $r = 4, 4, 4$ are changed to $r = 3, 4, 5$. We make the modification and train the models under the same setting as the baseline. The results, denoted as “Multigrid”, are

Table 2.4: Experimental results of models with the LargeFOV output layer but no MS-COCO pre-training on PASCAL VOC 2012 *val* set.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
DeepLabv2	92.8	84.1	37.9	82.9	65.2	**76.5**	89.9	82.7	87.9	33.2	74.9	50.2
G Interact	**93.0**	84.5	37.8	**84.2**	**66.5**	75.9	90.5	83.1	**88.4**	**34.6**	**75.4**	**52.3**
SS Conv	92.9	**85.5**	**38.1**	83.2	**66.5**	73.1	**91.2**	**84.0**	88.3	34.5	75.2	49.9

Models	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**mIoU**
DeepLabv2	80.6	76.6	78.6	82.1	52.2	77.4	40.8	80.1	66.6	71.1
G Interact	**81.7**	75.5	77.4	82.1	52.8	78.2	41.5	**81.7**	**67.9**	**71.7**
SS Conv	81.0	**77.2**	**79.5**	**82.5**	**53.7**	**78.6**	**42.0**	80.0	67.7	71.6

shown in the second lines of Tables 2.1 and 2.2. Surprisingly, our implementation indicates that the approach does not improve the performance. An explanation of the results is that the method should be applied together with other modifications, as both [38] and [37] conduct experiments together with other changes over DeepLabv2, such as dense upsamling convolution (DUC) and deeper encoders.

As we address the gridding artifacts in the last two blocks of the encoder, we also run experiments with different output layers in order to make the comparisons independent of the output layer. We replace the original atrous spatial pyramid pooling (ASPP) output layer of DeepLabv2 by the large field of view (LargeFOV) layer, which was applied earlier in [36]. We train the models with the same settings above, with and without MS-COCO pre-training, and show the results in Tables 2.3 and 2.4, respectively. Again, the proposed degridding methods result in significant improvements consistently.

2.4.3 Cityscapes

We further compare our proposed methods on the Cityscapes dataset [59]. Cityscapes collects $5,000$ 2048×1024 images of street scenes from 50 different cities and provides high quality pixel-wise annotations of 19 classes. The $5,000$ images are divided into *train*, *val* and *test* with $2,975$, 500 and $1,525$ images, respectively. Again, we train models on the *train* set and perform evaluation

Table 2.5: Experimental results of models with the ASPP output layer and MS-COCO pre-training on Cityscapes *val* set. Class $1-19$ represent "road, sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, bicycle", respectively. This is the same for Tables 2.5 and 2.6.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**mIoU**
DeepLabv2	97.2	**79.7**	90.1	47.4	49.2	50.3	57.3	69.0	**90.6**	59.8	**92.8**	75.9	55.6	92.5	67.5	80.5	64.8	59.7	71.7	71.1
G Interact	**97.3**	79.6	90.2	50.4	49.9	**50.5**	**58.5**	69.1	90.5	58.7	92.7	75.9	55.4	92.5	70.9	80.2	65.0	**60.6**	**71.8**	71.6
SS Conv	97.2	**79.7**	**90.3**	**51.1**	**50.5**	50.2	58.1	**69.3**	90.5	**60.0**	92.7	**76.1**	**55.9**	**92.7**	**72.7**	**81.9**	**66.0**	59.7	**71.8**	**71.9**

on the *val* set. The training batch size is 3, where each batch contains randomly cropped patches of size 571×571. The initial learning rates for all models are set to 0.0005. All the other settings are the same as those in Section 2.4.2.

Experiments are still conducted under both settings, *i*.e., with and without MS-COCO pre-training, and the results are given in Tables 2.5 and 2.6, respectively. We can see that both of the proposed methods increase the mIoU over the baseline, which shows that the improvements are independent of datasets.

2.4.4 Effective Receptive Field Analysis

Since we are addressing the gridding artifacts, we perform the effective receptive field (ERF) analysis [54, 39] to visualize the smoothing effect of our methods. These experiments further verify that the improvements of the proposed methods come from degridding. Given a block in DCNNs, the ERF analysis is an approach to characterize how much each unit in the input of the block affects a particular output unit of the block mathematically [54], instead of theoretically.

Following the steps in [54, 39], we analyze the models on PASCAL VOC 2012, with the ASPP output layer and MS-COCO pre-training. We compute the ERF for chosen blocks of the baseline and both of the proposed methods. Specifically, suppose the input and output feature maps of a

Table 2.6: Experimental results of models with the ASPP output layer but no MS-COCO pre-training on Cityscapes *val* set.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**mIoU**
DeepLabv2	97.0	77.9	89.4	44.6	48.6	48.7	54.1	66.7	**90.3**	**58.0**	**92.5**	73.9	51.9	91.6	**59.9**	75.5	60.5	56.3	69.6	68.8
G Interact	**97.1**	**78.7**	**89.6**	44.7	49.2	48.6	54.2	**67.0**	**90.3**	57.6	92.1	**74.3**	**52.2**	91.7	59.0	**77.1**	60.5	56.8	**70.1**	69.0
SS Conv	97.0	78.3	**89.6**	**45.2**	**49.4**	**48.9**	**54.6**	66.5	90.2	57.1	92.0	74.1	52.1	**91.8**	59.5	76.8	**63.5**	**58.8**	69.7	**69.2**

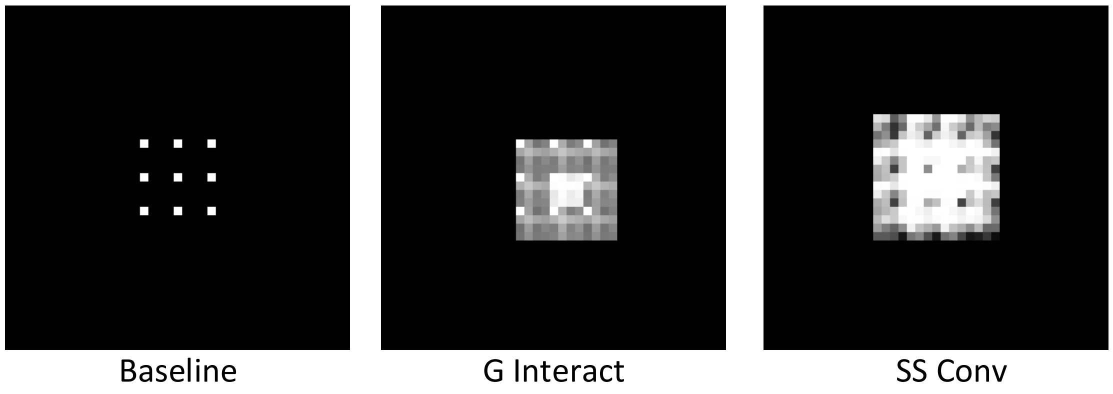


Figure 2.11: ERF visualization for the single dilated convolution with a kernel size of 3×3 and a dilation factor of $r = 4$. Black pixels represent zero weights.

block are x and y, respectively. The spatial locations of the feature maps are indexed by (i, j) with $(0, 0)$ representing the center. The ERF is measured by the partial derivative $\partial y_{0,0}/\partial x_{i,j}$. To compute it without an explicit loss function, we set the error gradient with respect to $y_{0,0}$ to 1 while for $y_{i,j}$ with $i \neq 0$ or $j \neq 0$, we set it to 0. Then the error gradient can be back-propagated to x and the error gradient with respect to $x_{i,j}$ equals to $\partial y_{0,0}/\partial x_{i,j}$ [54]. However, the results are input-dependent. So $\partial y_{0,0}/\partial x_{i,j}$ are computed for all images in the *val* set and their absolute values are averaged. Finally, we sum the values over all channels of x to get a visualization of the ERF.

In our experiments, we choose two blocks of the DCNNs to visualize the smoothing effect and

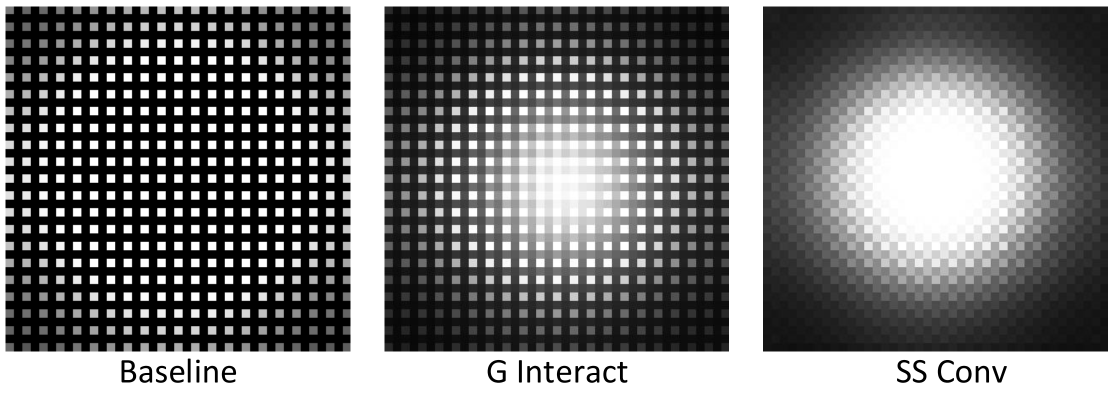


Figure 2.12: ERF visualization for the entire dilated convolutional block. Note that only the leftmost map has black pixels that represent zero weights.

enlarge the spatial size of visualizations ten times for display. The first block is the very last layer of the encoder, which is a dilated convolution with a kernel size of 3×3 and a dilation rate of $r = 4$. The ERF analysis results are presented in Figure 2.11. The ERF of the original dilated convolution in the baseline is obvious. It corresponds to a 3×3 filter with zeros inserted between non-zero weights. Such a filter results in the gridding problem. For our proposed degridding methods, we can see that they smooth the ERF and thus perform degridding. In addition, both methods expand the rectangular size of the ERF due to the SS operators. The second chosen block is the entire block composed of dilated convolutional layers, which includes the last two blocks of the encoder. Figure 2.12 shows the ERF visualization. The gridding artifacts are clearly smoothed in both proposed methods. In fact, only the leftmost visualization for the baseline has black pixels that represent zero weights. Particularly, we note that "SS FC" still has a grid-like visualization. A reason of this is the block-wise operation may result in larger grids in terms of blocks. Nevertheless, it alleviates the inconsistency of pixel-wise local information and improves DCNNs with dilated convolutions.

2.4.5 Separable and Shared Output Layer

We evaluate the proposed separable and shared (SS) output layer in Section 2.3.6 by only replacing the output layer of DeepLabv2. In our experiments, we set d_k and d_o to 512 in Equations 2.6

Table 2.7: Experimental results of models with the proposed SS output layer in Section 2.3.6 and MS-COCO pre-training on PASCAL VOC 2012 *val* set. "SS Output (#)" denotes the model using an SS output layer with a window size of $\# \times \#$. "SS Output Global" means the window size is chosen to be larger than the spatial sizes of inputs.

Models	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**mIoU**
DeepLabv2 (LargeFOV)	93.7	85.7	39.4	85.9	67.6	79.0	93.1	86.0	90.7	36.2	79.8	54.6	83.7	80.9	81.4	85.0	57.5	83.5	45.5	84.5	74.1	74.7
DeepLabv2 (ASPP)	93.8	85.9	38.8	84.8	64.3	79.0	93.7	85.5	91.7	34.1	83.0	57.0	86.1	83.0	81.0	85.0	58.2	83.4	48.2	87.2	74.0	75.1
SS Output (15)	94.1	86.5	39.0	86.2	65.9	80.3	93.8	87.4	90.7	36.0	82.1	59.2	84.2	80.8	81.2	85.5	58.3	84.1	48.1	87.4	74.1	75.5
SS Output (20)	94.1	86.7	40.2	86.9	66.2	**80.5**	94.5	**87.9**	91.7	36.1	83.6	**60.0**	86.2	83.9	82.5	85.6	58.9	84.0	48.9	88.7	74.3	76.3
SS Output (30)	94.2	87.9	**40.9**	**87.3**	66.1	79.7	**94.8**	**87.9**	92.9	36.5	84.9	59.6	**88.3**	86.8	**83.9**	**85.7**	**59.9**	84.2	50.0	89.5	**74.9**	77.0
SS Output Global	**94.4**	**89.2**	40.6	84.9	**69.7**	78.9	94.7	86.8	**93.2**	**38.1**	**89.9**	59.4	87.8	**87.3**	82.6	**85.7**	59.4	**89.5**	**52.4**	**90.2**	74.7	**77.6**

Table 2.8: Comparison of the number of training parameters between different output layers. The numbers of channels in inputs and outputs are set to $2,048$ and 512, respectively. Note that the SS output layer has the same number of parameters when changing the window size.

Models	#Parameters
LargeFOV	9,437,696
ASPP	37,750,784
SS Output	**3,409,920**

to 2.8 and use 8 heads in the graph attention mechanism. Different window sizes of the SS output layer are explored. Table 2.7 provides the comparison results between the original DeepLabv2 and models using SS output layers. Clearly, the SS output layer shows its effectiveness by improving the performance significantly. It is worth noting that the larger the window size is, the more the performance gets improved, which indicates the importance of aggregating global information for

prediction.

In order to show the efficiency of our SS output layer, we also compare the number of training parameters between different output layers in Table 2.8. The number of input channels to the output layer is set to $2,048$. To be fair, we only compute the number of parameters of the operators before the 1×1 regular convolutions and set the number of output channels of the operators to 512. In this case, the LargeFov output layer has a single 3×3 dilated convolution and the ASPP output layer has four 3×3 dilated convolutions, while the SS output layer contains the proposed SS operation in Section 2.3.6 with $d_k = d_o = 512$. Note that the SS output layer has the same number of parameters when changing the window size. According to Table 2.8, the proposed SS output layer reduces a large amount of training parameters as compared with output layers based on dilated convolutions.

3. DEEP ATTENTION NETWORKS FOR IMAGES: NON-LOCAL U-NETS FOR BIOMEDICAL IMAGE SEGMENTATION

As indicated by Section 2.4.5 of Chapter 2, aggregating global information is important for dense prediction tasks, and the attention mechanism is able to perform such global aggregation. In this chapter, we get motivated by this finding and propose non-local U-Nets for biomedical image segmentation tasks. In particular, we extend the attention mechanism to a global aggregation block that can be flexibly used as size-preserving, down-sampling, and up-sampling layers.*

3.1 Introduction

In recent years, deep learning methods, such as fully convolutional networks (FCN) [51], U-Net [63], Deeplab [36, 27], and RefineNet [64], have continuously set performance records on image segmentation tasks. In particular, U-Net has served as the backbone network for biomedical image segmentation. Basically, U-Net is composed of a down-sampling encoder and an up-sampling decoder, along with skip connections between them. It incorporates both local and global contextual information through the encoding-decoding process.

Many variants of U-Net have been developed and they achieved improved performance on biomedical image segmentation tasks. For example, residual deconvolutional network [65] and residual symmetric U-Net [66] addressed the 2D electron microscopy image segmentation task by building a U-Net based network with additional short-range residual connections [10]. In addition, U-Net was extended from 2D to 3D cases for volumetric biomedical images, leading to models like 3D U-Net [67], V-Net [68], and convolution-concatenate 3D-FCN (CC-3D-FCN) [69].

Despite the success of these studies, we conduct an in-depth study of U-Net based models and observe two limitations shared by them. First, the encoder usually stacks size-preserving convolutional layers, interlaced with down-sampling operators, to gradually reduce the spatial sizes

*Reprinted with permission from "Non-local U-Nets for biomedical image segmentation." by Zhengyang Wang, Na Zou, Dinggang Shen, and Shuiwang Ji, 2020, *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, vol. 1, pp. 6315-6322, Copyright 2020 by AAAI.

of feature maps. Both convolutions and down-sampling operators are typically local operators, which apply small kernels to scan inputs and extract local information. Stacking them in a cascade way results in large effective kernels and is able to aggregate long-range information. As the biomedical image segmentation usually benefits from a wide range of contextual information, most prior models have a deep encoder, *i.e.,* an encoder with many stacked local operators. It hurts the efficiency of these models by introducing a considerably large amount of training parameters, especially when more down-sampling operators are employed, since the number of feature maps usually gets doubled after each down-sampling operation. In addition, more down-sampling operators cause the loss of more spatial information during encoding, which is crucial for biomedical image segmentation. Second, the decoder is built in a similar way to the encoder, by replacing down-sampling operators with up-sampling operators. Popular up-sampling operators, like deconvolutions and unpooling layers, are local operators as well [41]. However, the up-sampling process involves the recovery of spatial information, which is hard without taking global information into consideration. To conclude, it will improve both the effectiveness and efficiency of U-Net based models to develop a new operator capable of performing non-local information aggregation. As U-Net has size-preserving processes, as well as down-sampling and up-sampling layers, the new operator is supposed to be flexible to fit these cases.

In this chapter, we address the two limitations and propose the non-local U-Nets for biomedical image segmentation. To address the first limitation above, we propose a global aggregation block based on the self-attention operator [13, 70, 71], which is able to aggregate global information without a deep encoder. This block is further extended to an up-sampling global aggregation block, which can alleviate the second problem. To the best of our knowledge, we are the first to make this extension. We explore the applications of these flexible global aggregation blocks in U-Net on the 3D multimodality isointense infant brain magnetic resonance (MR) image segmentation task. Experimental results show that our proposed non-local U-Nets are able to achieve the top performance with fewer parameters and faster computation.

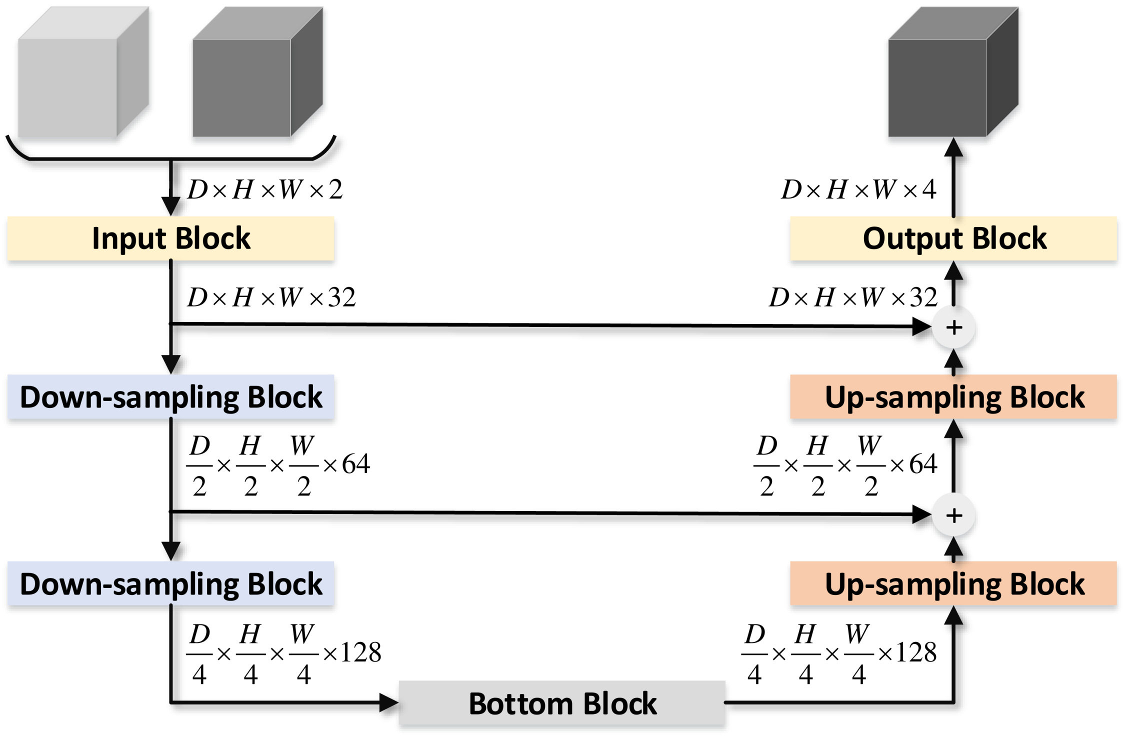

$D \times H \times W \times 2$
Input Block
$D \times H \times W \times 32$
Down-sampling Block
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 64$
Down-sampling Block
$\frac{D}{4} \times \frac{H}{4} \times \frac{W}{4} \times 128$
Bottom Block
$\frac{D}{4} \times \frac{H}{4} \times \frac{W}{4} \times 128$
Up-sampling Block
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 64$
+
Up-sampling Block
$D \times H \times W \times 32$
+
Output Block
$D \times H \times W \times 4$

Figure 3.1: An illustration of the U-Net framework employed by our proposed non-local U-Nets. In this example, the inputs have 2 channels and the segmentation task has 4 classes.

3.2 Non-local U-Nets

In this section, we introduce our proposed non-local U-Nets. We first illustrate the specific U-Net framework used by our models. Based on the framework, our models are composed of different size-preserving, down-sampling and up-sampling blocks. We describe each block and propose our global aggregation blocks to build the non-local U-Nets.

3.2.1 U-Net Framework

We describe the non-local U-Nets in 3D cases. Lower or higher dimensional cases can be easily derived. An illustration of the basic U-Net framework is given in Fig. 3.1. The input first goes through an encoding input block, which extracts low-level features. Two down-sampling blocks are used to reduce the spatial sizes and obtain high-level features. Note that the number of channels is doubled after each down-sampling block. A bottom block then aggregates global information and produces the output of the encoder. Correspondingly, the decoder uses two up-sampling blocks to recover the spatial sizes for the segmentation output. The number of feature maps is halved after an up-sampling operation.

To assist the decoding process, skip connections copy feature maps from the encoder to the decoder. Differently, in the non-local U-Nets, the copied feature maps are combined with decoding feature maps through summation, instead of concatenation used in U-Net [63, 72]. The intuitive way to combine features from the encoder and the decoder is concatenation, providing two sources of inputs to the up-sampling operation. Using summation instead has two advantages [73]. First, summation does not increase the number of feature maps, thus reducing the number of trainable parameters in the following layer. Second, skip connections with summation can be considered as long-range residual connections, which are known to be capable of facilitating the training of models.

Given the output of the decoder, the output block produces the segmentation probability map. Specifically, for each voxel, the probabilities that it belongs to each segmentation class are provided, respectively. The final segmentation map can be obtained through a single *argmax* operation on this probability map. The details of each block are introduced in following sections.

3.2.2 Residual Blocks

Residual connections have been shown to facilitate the training of deep learning models and achieve better performance [10]. Note that skip connections with summation in our U-Net framework are equivalent to long-range residual connections. To further improve U-Net, the studies in [66, 64, 65] proposed to add short-range residual connections as well. However, those studies did not apply residual connections for down-sampling and up-sampling blocks. Down-sampling block with residual connections has been explored in ResNet [10]. We explore the idea for up-sampling blocks based on our proposed up-sampling global aggregation block, as discussed in next section.

In our proposed model, four different residual blocks are used to form a fully residual network, as shown in Fig. 3.2. Notably, all of them apply the pre-activation pattern [74]. Fig. 3.2(a) shows

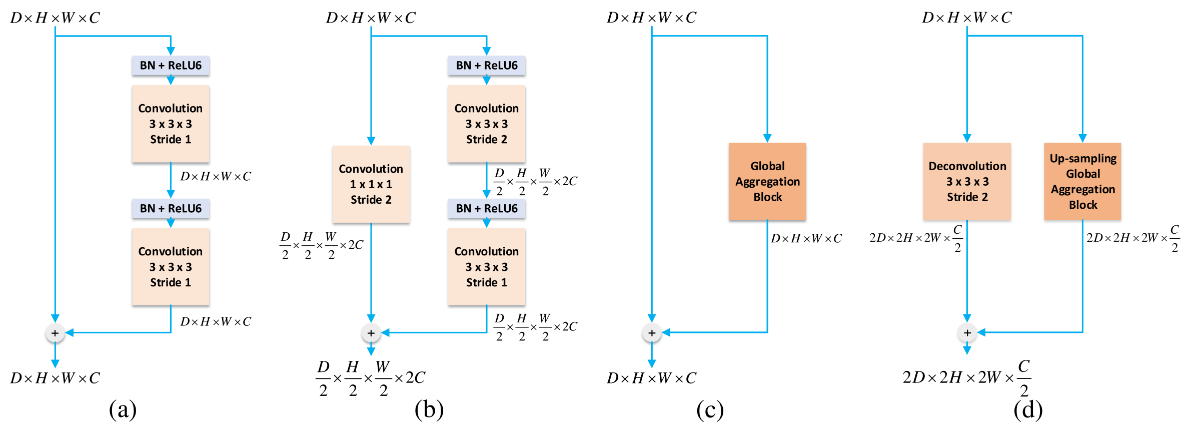

$D \times H \times W \times C$
BN + ReLU6
Convolution 3 x 3 x 3 Stride 1
$D \times H \times W \times C$
BN + ReLU6
Convolution 3 x 3 x 3 Stride 1
$D \times H \times W \times C$
$D \times H \times W \times C$
(a)
$D \times H \times W \times C$
BN + ReLU6
Convolution 3 x 3 x 3 Stride 2
Convolution 1 x 1 x 1 Stride 2
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 2C$
BN + ReLU6
Convolution 3 x 3 x 3 Stride 1
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 2C$
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 2C$
$\frac{D}{2} \times \frac{H}{2} \times \frac{W}{2} \times 2C$
(b)
$D \times H \times W \times C$
Global Aggregation Block
$D \times H \times W \times C$
$D \times H \times W \times C$
(c)
$D \times H \times W \times C$
Deconvolution 3 x 3 x 3 Stride 2
Up-sampling Global Aggregation Block
$2D \times 2H \times 2W \times \frac{C}{2}$
$2D \times 2H \times 2W \times \frac{C}{2}$
$2D \times 2H \times 2W \times \frac{C}{2}$
(d)

Figure 3.2: An illustration of the residual blocks employed by our proposed non-local U-Nets. Details are provided in Section "Residual Block".

a regular residual block with two consecutive convolutional layers. Here, batch normalization [75] with the ReLU6 activation function is used before each convolutional layer. This block is used as the input block in our framework. The output block is constructed by this block followed by a $1 \times 1 \times 1$ convolution with a stride of 1. Moreover, after the summation of skip connections, we insert one such block. Fig. 3.2(b) is a down-sampling residual block. A $1 \times 1 \times 1$ convolution with a stride of 2 is used to replace the identity residual connection, in order to adjust the spatial sizes of feature maps accordingly. We employ this block as the down-sampling blocks. Fig. 3.2(c) illustrates our bottom block. Basically, a residual connection is applied on the proposed global aggregation block. The up-sampling residual block is provided in Fig. 3.2(d). Similar to the down-sampling block in Fig. 3.2(b), the identity residual connection is replaced by a $3 \times 3 \times 3$ deconvolution with a stride of 2 and the other branch is the up-sampling global aggregation block. Our model uses this block as the up-sampling blocks.

3.2.3 Global Aggregation Block

To achieve global information fusion through a block, each position of the output feature maps should depend on all positions of the input feature maps. Such an operation is opposite to local operations like convolutions and deconvolutions, where each output location has a local receptive field on the input. In fact, a fully-connected layer has this global property. However, it is prone to over-fitting and does not work well in practice. We note that the self-attention block used in the Transformer [13] computes outputs at one position by attending to every position of the input. Later, the study in [70] proposed non-local neural networks for video classification, which employed a similar block. While both studies applied self-attention blocks with the aim of capturing long-term dependencies in sequences, we point out that global information of image feature maps can be aggregated through self-attention blocks.

Based on this insight, we propose the global aggregation block, which is able to fuse global information from feature maps of any size. We further generalize it to handle down-sampling and up-sampling, making it a block that can be used anywhere in deep learning models.

Let X represent the input to the global aggregation block and Y represent the output. For simplicity, we use $Conv\_1_N$ to denote a $1 \times 1 \times 1$ convolution with a stride of 1 and N output channels. Note that $Conv\_1_N$ does not change the spatial size. The first step of the proposed block is to generate the query (Q), key (K) and value (V) matrices [13], given by

$$\begin{aligned} Q &= Unfold(QueryTransform_{C_K}(X)), \\ K &= Unfold(Conv\_1_{C_K}(X)), \\ V &= Unfold(Conv\_1_{C_V}(X)), \end{aligned} \tag{3.1}$$

where $Unfold(\cdot)$ unfolds a $D \times H \times W \times C$ tensor into a $(D \times H \times W) \times C$ matrix, and C_K, C_V are hyper-parameters representing the dimensions of the keys and values. $QueryTransform_{C_K}(\cdot)$ can be any operation that produces C_K feature maps. Suppose the size of X is $D \times H \times W \times C$. Then the dimensions of K and V are $(D \times H \times W) \times C_K$ and $(D \times H \times W) \times C_V$, respectively. The dimension of Q, however, is $(D_Q \times H_Q \times W_Q) \times C_K$, where D_Q, H_Q, W_Q depend on $QueryTransform(\cdot)$. The left part of Fig. 3.3 illustrates this step. Here, a $D \times H \times W \times C$ tensor is represented by a $D \times H \times W$ cube, whose voxels correspond to C-dimensional vectors.

Each row of the Q, K and V matrices denotes a query vector, a key vector and a value vector, respectively. Note that the query vector has the same dimension as the key vector. Meanwhile,

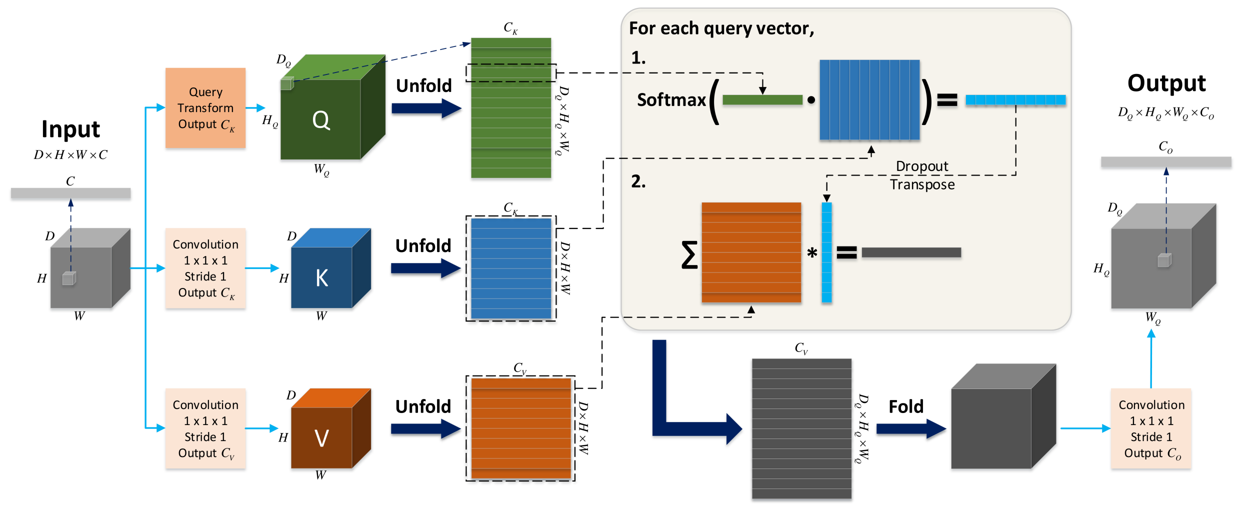

Input
$D \times H \times W \times C$
Query Transform Output C_K
Convolution 1 x 1 x 1 Stride 1 Output C_K
Convolution 1 x 1 x 1 Stride 1 Output C_V
Q
K
V
Unfold
Unfold
Unfold
For each query vector,
1.
Softmax
2.
Dropout
Transpose
Fold
Output
$D_Q \times H_Q \times W_Q \times C_O$
Convolution 1 x 1 x 1 Stride 1 Output C_O

Figure 3.3: An illustration of our proposed global aggregation block. Note that the spatial size of the output is determined by that of the query (Q) matrix.

the number of key vectors is the same as that of value vectors, which indicates a one-to-one correspondence. In the second step, the attention mechanism is applied on Q, K and V [13], defined as

$$\begin{aligned} A &= Softmax(\frac{QK^T}{\sqrt{C_K}}), \\ O &= AV, \end{aligned} \tag{3.2}$$

where the dimension of the attention weight matrix A is $(D_Q \times H_Q \times W_Q) \times (D \times H \times W)$ and the dimension of the output matrix O is $(D_Q \times H_Q \times W_Q) \times C_V$. To see how it works, we take one query vector from Q as an example. In the attention mechanism, the query vector interacts with all key vectors, where the dot-product between the query vector and one key vector produces a scalar weight for the corresponding value vector. The output of the query vector is a weighted sum of all value vectors, where the weights are normalized through $Softmax$. This process is repeated for all query vectors and generates $(D_Q \times H_Q \times W_Q)$ C_V-dimensional vectors. This step is illustrated in the box of Fig. 3.3. Note that Dropout [76] can be applied on A to avoid over-fitting. As shown in Fig. 3.3, the final step of the block computes Y by

$$Y = Conv\_1_{C_O}(Fold(O)), \tag{3.3}$$

where $Fold(\cdot)$ is the reverse operation of $Unfold(\cdot)$ and C_O is a hyper-parameter representing the dimension of the outputs. As a result, the size of Y is $D_Q \times H_Q \times W_Q \times C_O$.

In particular, it is worth noting that the spatial size of Y is determined by that of the Q matrix, *i.e.*, by the $QueryTransform_{C_K}(\cdot)$ function in (3.1). Therefore, with appropriate functions, the global aggregation block can be flexibly used for size-preserving, down-sampling and up-sampling processes. In our proposed non-local U-Nets, we set $C_K = C_V = C_O$ and explore two different $QueryTransform_{C_K}(\cdot)$ functions. For the global aggregation block in Fig. 3.2(c), $QueryTransform_{C_K}(\cdot)$ is $Conv\_1_{C_K}$. And for the up-sampling global aggregation block in Fig. 3.2(d), $QueryTransform_{C_K}(\cdot)$ is a $3 \times 3 \times 3$ deconvolution with a stride of 2. The use of this block alleviates the problem that the up-sampling through a single deconvolution loses information. By taking global information into consideration, the up-sampling block is able to recover more accurate details.

3.3 Experimental Studies

We perform experiments on the 3D multimodality isointense infant brain MR image segmentation task to evaluate our non-local U-Nets. The task is to perform automatic segmentation of MR images into cerebrospinal fluid (CSF), gray matter (GM) and white matter (WM) regions. We first introduce the baseline model and the evaluation methods used in our experiments. Then the training and inference processes are described. We provide comparison results in terms of both effectiveness and efficiency, and conduct ablation studies to demonstrate that how each global aggregation block in our non-local U-Nets improves the performance. In addition, we explore the trade-off between the inference speed and accuracy based on different overlapping step sizes, and analyze the impact of patch size. The experimental code and dataset information have been made publicly available [†].

3.3.1 Experimental Setup

We use CC-3D-FCN [69] as our baseline. CC-3D-FCN is a 3D fully convolutional network (3D-FCN) with convolution and concatenate (CC) skip connections, which is designed for 3D multimodality isointense infant brain image segmentation. It has been shown to outperform traditional machine learning methods, such as FMRIB's automated segmentation tool (FAST) [77], majority voting (MV), random forest (RF) [78] and random forest with auto-context model (LINKS) [79]. Moreover, studies in [69] has showed the superiority of CC-3D-FCN to previous deep learning models, like 2D, 3D CNNs [80], DeepMedic [81], and the original 3D U-Net [67]. Therefore, it is appropriate to use CC-3D-FCN as the baseline of our experiments. Note that our dataset is different from that in [69].

In our experiments, we employ the Dice ratio (DR) and propose the 3D modified Hausdorff distance (3D-MHD) as the evaluation metrics. These two methods evaluate the accuracy only for binary segmentation tasks, so it is required to transform the 4-class segmentation map predicted by our model into 4 binary segmentation maps for evaluation. That is, a 3D binary segmentation map should be constructed for each class, where 1 denotes the voxel in the position belongs to the class and 0 means the opposite. In our experiments, we derive binary segmentation maps directly from 4-class segmentation maps. The evaluation is performed on binary segmentation maps for CSF, GM and WM.

Specifically, let P and L represent the predicted binary segmentation map for one class and the corresponding ground truth label, respectively. The DR is given by $DR = 2|P \cap L|/(|P| + |L|)$, where $|\cdot|$ denotes the number of 1's in a segmentation map and $|P \cap L|$ means the number of 1's shared by P and L. Apparently, DR is a value in $[0, 1]$ and a larger DR indicates a more accurate segmentation.

The modified Hausdorff distance (MHD) [82] is designed to compute the similarity between two objects. Here, an object is a set of points where a point is represented by a vector. Specifically,

[†]`https://github.com/divelab/Non-local-U-Nets`

given two sets of vectors A and B, MHD is computed by $MHD = \max(d(A, B), d(B, A))$, where the distance between two sets is defined as $d(A, B) = 1/|A| \sum_{a \in A} d(a, B)$, and the distance between a vector and a set is defined as $d(a, B) = \min_{b \in B} ||a - b||$. Previous studies [79, 80, 69] applied MHD for evaluation by treating a 3D $D \times H \times W$ map as $H \times W$ D-dimensional vectors. However, there are two more different ways to vectorize the 3D map, depending on the direction of forming vectors, *i.e.,* $D \times H$ W-dimensional vectors and $D \times W$ H-dimensional vectors. Each vectorization leads to different evaluation results by MHD. To make it a direction-independent evaluation metric as DR, we define 3D-MHD, which computes the averaged MHD based on the three different vectorizations. A smaller 3D-MHD indicates a higher segmentation accuracy.

3.3.2 Training and Inference Strategies

Our proposed non-local U-Nets apply Dropout [76] with a rate of 0.5 in each global aggregation block and the output block before the final $1 \times 1 \times 1$ convolution. A weight decay [83] with a rate of $2e - 6$ is also employed. To train the model, we use randomly cropped small patches. In this way, we obtain sufficient training data and the requirement on memory is reduced. No extra data augmentation is needed. The experimental results below suggest that patches with a size of 32^3 leads to the best performance. The batch size is set to 5. The Adam optimizer [84] with a learning rate of 0.001 is employed to perform the gradient descent algorithm.

In the inference process, following [69], we extract patches with the same size as that used in training. For example, to generate 32^3 patches for inference, we slide a window of size 32^3 through the original image with a constant overlapping step size. The overlapping step size must be smaller than or equal to the patch size, in order to guarantee that extracted patches cover the whole image. Consequently, prediction for all these patches provides segmentation probability results for every voxel in the original image. For voxels that receive multiple results due to overlapping, we average them to produce the final prediction. The overlapping step size is an important hyper-parameter affecting the inference speed and the segmentation accuracy. A smaller overlapping step size results in better accuracy, but increases the inference time as more patches are generated. We explore the trade-off in our experiments.

3.3.3 Comparison with the Baseline

Table 3.1: Comparison of segmentation performance between our proposed model and the baseline model in terms of DR. The leave-one-subject-out cross-validation is used. Larger values indicate better performance.

Model	CSF	GM	WM	Average
Baseline	0.9250±0.0118	0.9084±0.0056	0.8926±0.0119	0.9087±0.0066
Non-local U-Net	**0.9530±0.0074**	**0.9245±0.0049**	**0.9102±0.0101**	**0.9292±0.0050**

Table 3.2: Comparison of segmentation performance between our proposed model and the baseline model in terms of 3D-MHD. The leave-one-subject-out cross-validation is used. Smaller values indicate better performance. Note that 3D-MHD gives different results from MHD.

Model	CSF	GM	WM	Average
Baseline	0.3417±0.0245	0.6537±0.0483	0.4817±0.0454	0.4924±0.0345
Non-local U-Net	**0.2554±0.0207**	**0.5950±0.0428**	**0.4454±0.0040**	**0.4319±0.0313**

We compare our non-local U-Nets with the baseline on our dataset. Following [69], the patch size is set to 32^3 and the overlapping step size for inference is set to 8. To remove the bias of different subjects, the leave-one-subject-out cross-validation is used for evaluating segmentation performance. That is, for 10 subjects in our dataset, we train and evaluate models 10 times correspondingly. Each time one of the 10 subjects is left out for validation and the other 9 subjects are used for training. The mean and standard deviation of segmentation performance of the 10 runs are reported.

Tables 3.1 and 3.2 provide the experimental results. In terms of both evaluation metrics, our non-local U-Nets achieve significant improvements over the baseline model. Due to the small variances of the results, we focus on one of the 10 runs for visualization and ablation studies, where

Table 3.3: Comparison of the number of parameters between our proposed model and the baseline model.

Model	Number of Parameters
Baseline	2,534,276
Non-local U-Net	**1,821,124**

the models are trained on the first 9 subjects and evaluated on the 10^{th} subject. A visualization of the segmentation results in this run is given by Fig. 3.4. By comparing the areas in red circles, we can see that our model is capable of catching more details than the baseline model. We also visualize the training processes to illustrate the superiority of our model. Fig. 3.5 shows the training and validation curves in this run of our model and the baseline model, respectively. Clearly, our model converges faster to a lower training loss. In addition, according to the better validation results, our model does not suffer from over-fitting.

To further show the efficiency of our proposed model, we compare the number of parameters as reported in Table 3.3. Our model reduces 28% parameters compared to CC-3D-FCN and achieves better performance. A comparison of inference time is also provided in Table 3.4. The settings of our device are - GPU: Nvidia Titan Xp 12GB; CPU: Intel Xeon E5-2620v4 2.10GHz; OS: Ubuntu 16.04.3 LTS.

Since our data has been used as the training data in the iSeg-2017 challenge, we also compare the results evaluated on the 13 testing subjects in Table 3.5. According to the leader board, our model achieves one of the top performances. Results in terms of DR are reported since it is the only shared evaluation metric.

3.3.4 Ablation Studies of Different Modules

We perform ablation studies to show the effectiveness of each part of our non-local U-Nets. Specifically, we compare the following models:

Model1 is a 3D U-Net without short-range residual connections. Down-sampling and up-sampling are implemented by convolutions and deconvolutions with a stride of 2, respectively.

Table 3.4: Comparison of inference time between our proposed model and the baseline model. The leave-one-subject-out cross-validation is used. The patch size is set to 32^3 and the overlapping step size for inference is set to 8.

Model	Inference Time (min)
Baseline	3.85±0.15
Non-local U-Net	**3.06±0.12**

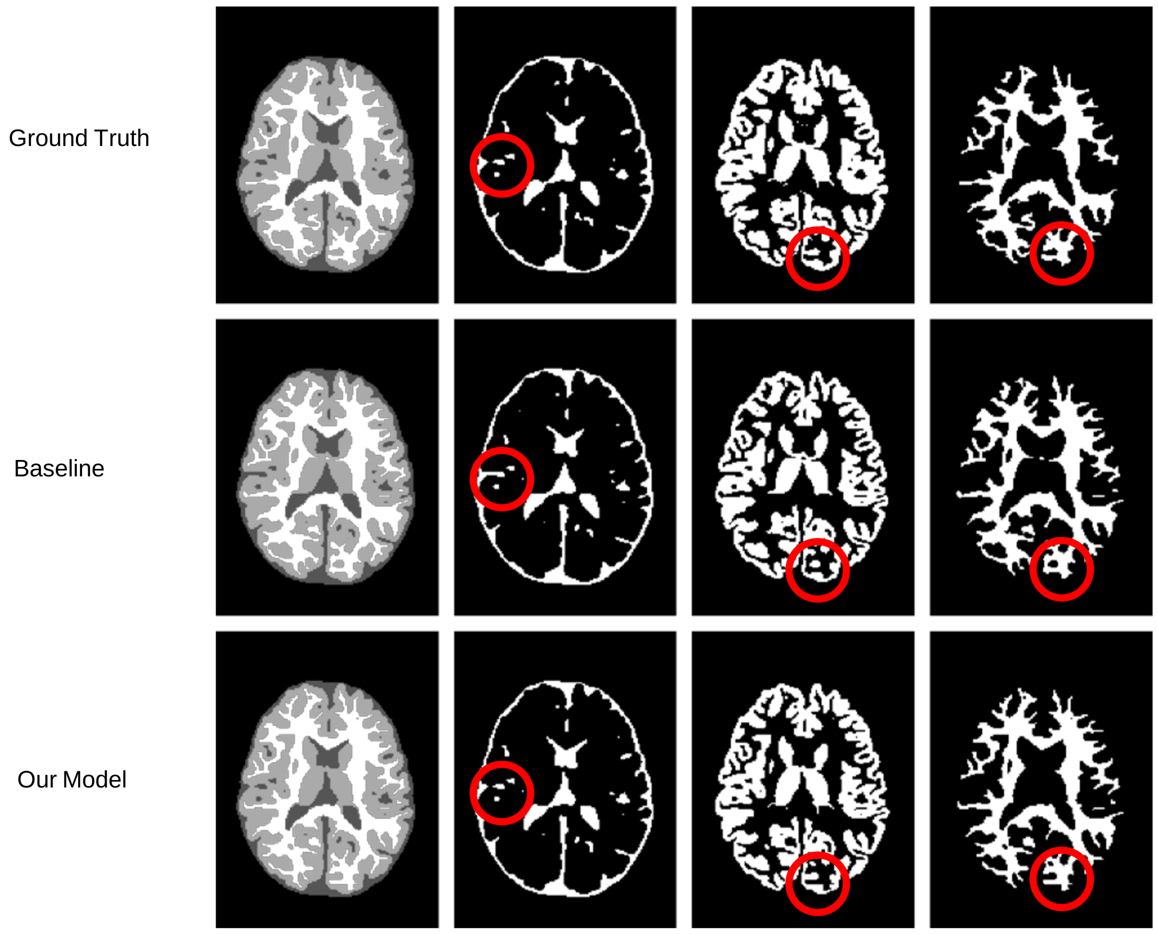


Figure 3.4: Visualization of the segmentation results on the 10^{th} subject by our proposed model and the baseline model. Both models are trained on the first 9 subjects. The first column shows the original segmentation maps. The second, third and fourth columns show the binary segmentation maps for CSF, GM and WM, respectively.

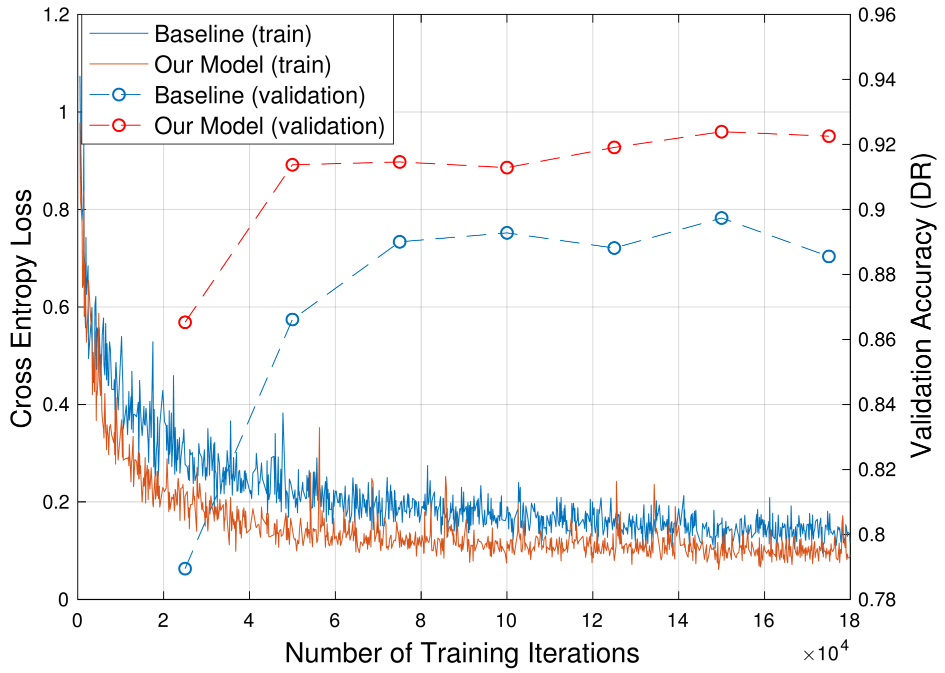


Figure 3.5: Comparison of training processes and validation results between our proposed model and the baseline model when training on the first 9 subjects and using the 10^{th} subject for validation.

Table 3.5: Comparison of segmentation performance on the 13 testing subjects of iSeg-2017 between our proposed model and the baseline model in terms of DR. Larger values indicate better performance.

Model	**CSF**	**GM**	**WM**
Baseline	0.9324±0.0067	0.9146±0.0074	0.8974±0.0123
Non-local U-Net	**0.9557±0.0060**	**0.9219±0.0089**	**0.9044±0.0153**

The bottom block is simply a convolutional layer. Note that the baseline model, CC-3D-FCN, has showed improved performance over 3D U-Net [69]. However, the original 3D U-Net was not designed for this task [67]. In our experiments, we appropriately set the hyperparameters of 3D U-Net and achieve better performance.

Model2 is Model1 with short-range residual connections, *i.e.*, the blocks in Fig. 3.2(a) and (b) are applied. The bottom block and up-sampling blocks are the same as those in Model1.

Table 3.6: Ablation study by comparing segmentation performance between different models in terms of DR. All models are trained on the first 9 subjects and evaluated on the 10^{th} subject. Larger values indicate better performance. Details of models are provided in the text.

Model	**CSF**	**GM**	**WM**	**Average**
Model1	**0.9585**	0.9099	0.8625	0.9103
Model2	0.9568	0.9172	0.8728	0.9156
Model3	0.9576	0.9198	0.8749	0.9174
Model4	0.9578	0.9210	0.8769	0.9186
Model5	0.9554	0.9225	0.8804	0.9194
Non-local U-Net	0.9572	**0.9278**	**0.8867**	**0.9239**

Table 3.7: Ablation study by comparing segmentation performance between different models in terms of 3D-MHD. All models are trained on the first 9 subjects and evaluated on the 10^{th} subject. Smaller values indicate better performance. Note that 3D-MHD gives different results from MHD. Details of models are provided in the text.

Model	**CSF**	**GM**	**WM**	**Average**
Model1	**0.2363**	0.6277	0.4705	0.4448
Model2	0.2404	0.6052	0.4480	0.4312
Model3	0.2392	0.5993	0.4429	0.4271
Model4	0.2397	0.5926	0.4336	0.4220
Model5	0.2444	0.5901	0.4288	0.4211
Non-local U-Net	0.2477	**0.5692**	**0.4062**	**0.4077**

Model3 replaces the first up-sampling block in Model2 with the block in Fig. 3.2(d).

Model4 replaces both up-sampling blocks in Model2 with the block in Fig. 3.2(d).

Model5 replaces the bottom block in Model2 with the block in Fig. 3.2(c).

All models are trained on the first 9 subjects. We report the segmentation performance on the 10^{th} subject in Table 3.6 and Table 3.7. The results demonstrate how different global aggregation blocks in our non-local U-Nets improve the performance.

3.3.5 Impact of the Overlapping Step Size

As discussed above, a small overlapping step size usually results in better segmentation, due to the ensemble effect. However, with a small overlapping step size, the model has to perform

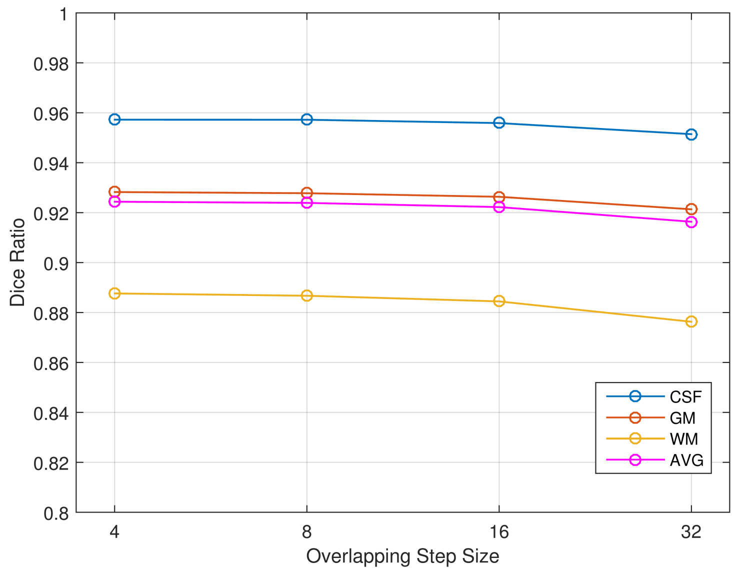


Figure 3.6: Changes of segmentation performance in terms of DR, with respect to different overlapping step sizes during inference. The model is trained on the first 9 subjects and evaluated on the 10^{th} subject.

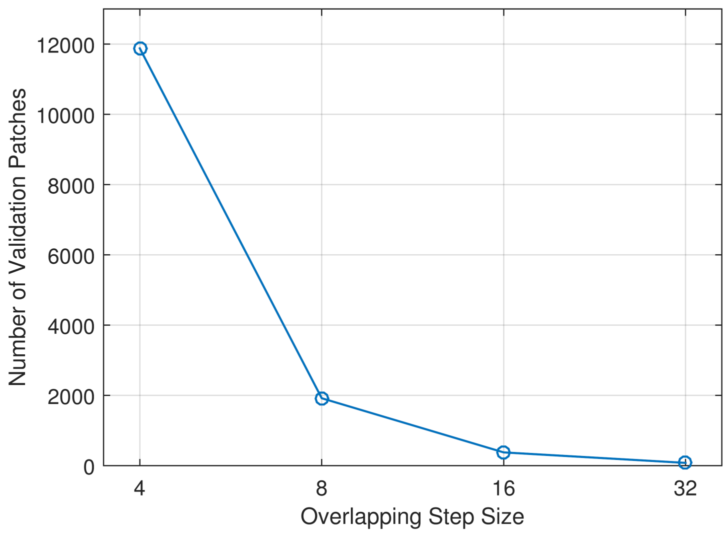


Figure 3.7: Changes of the number of validation patches for the 10^{th} subject, with respect to different overlapping step sizes during inference.

inference for more validation patches and thus decreases the inference speed. We explore the trade-off in our non-local U-Nets by setting the overlapping step sizes to 4, 8, 16, 32, respectively.

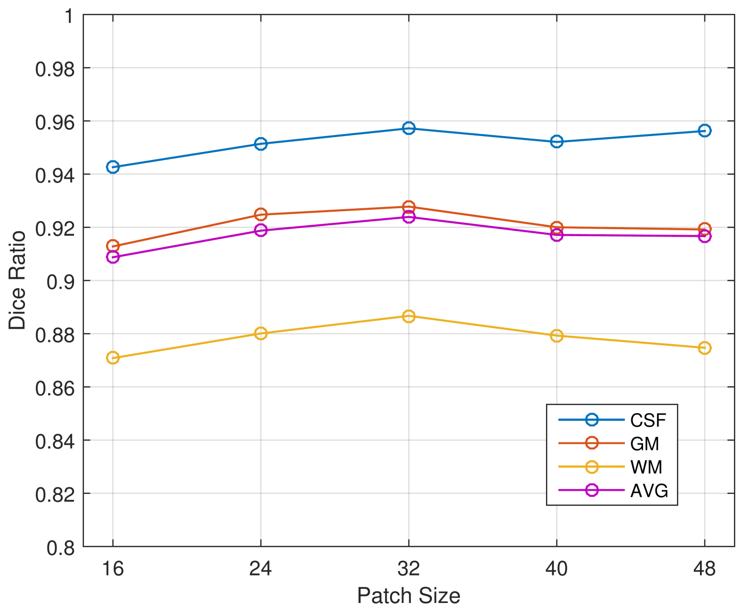


Figure 3.8: Changes of segmentation performance in terms of DR, with respect to different patch sizes. The model is trained on the first 9 subjects and evaluated on the 10^{th} subject.

Again, we train our model on the first 9 subjects and perform evaluation on the 10^{th} subject. The patch size is set to 32^3. According to the overlapping step sizes, 11880, 1920, 387, 80 patches need to be processed during inference, as shown in Fig. 3.7. In addition, Fig. 3.6 plots the changes of segmentation performance in terms of DR. Obviously, 8 and 16 are good choices that achieve accurate and fast segmentation results.

3.3.6 Impact of the Patch Size

The patch size affects the total number of distinct training samples. Meanwhile, it controls the range of available global information when performing segmentation for a patch. To choose the appropriate patch size for the non-local U-Nets, we perform a grid search by training on the first 9 subjects and evaluating on the 10^{th} subject with the overlapping step size of 8. Experiments are conducted with five different patch sizes: 16^3, 24^3, 32^3, 40^3, 48^3. The results are provided in Fig. 3.8, where 32^3 obtains the best performance and is selected as the default setting of our model.

4. DEEP ATTENTION NETWORKS FOR IMAGES: GLOBAL VOXEL TRANSFORMER NETWORKS FOR AUGMENTED MICROSCOPY

In this chapter, we extend the success of our non-local U-Nets to a wide range of augmented microscopy applications. In particular, we design more effective and efficient attention-based blocks named global voxel transformer operators (GVTOs) and develop an open-source tool named global voxel transformer networks (GVTNets). In addition, we also investigate the generalization ability of the attention mechanism under the transfer learning setting.

4.1 Introduction

In modern biology and life science, augmented microscopy attempts to improve the quality of microscope images to extract more information, such as introducing fluorescent labels, increasing the signal-to-noise ratio (SNR), and performing super-resolution. Previous advances in microscopy have allowed the imaging of biological processes with higher and higher quality [85, 86, 87, 88, 89, 90, 91, 92]. However, these advanced augmented-microscopy techniques usually lead to high costs in terms of the microscopy hardware and experimental conditions, resulting in many practical limitations. In addition, specific concerns are raised when recording processes of live cells, tissues, and organisms; those are, the imaging process should neither significantly affect the biological processes nor substantially harm the sample's health. For example, assessing phototoxicity is a major problem in live fluorescence imaging [93, 94]. With these restrictions, high-quality microscope images are hard, expensive, and slow to obtain. While some microscope images, like transmitted-light images [95], can be collected at relatively low cost, they are not sufficient to provide accurate statistics and correct insights without augmentation. As a result, modern biologists and life scientists usually have to deal with the trade-offs between the quality of microscope images and the restrictions in the process of collecting them [96, 97, 98].

In recent years, the development of deep learning [8] has pushed the boundaries of such trade-offs by enabling fast and inexpensive microscopy augmentation using computational approaches [99, 100, 101]. The augmented microscopy task is formulated as a biological image transformation problem in deep learning. Specifically, models composed of multi-layer artificial neural networks take low-quality microscope images as inputs, and transform them into high-quality ones through computational processes. Deep learning has led to success in various augmented microscopy applications, such as prediction of fluorescence signals from transmitted-light images [102, 103, 104, 72, 105, 106, 107, 92], virtual refocusing of fluorescence images [108], content-aware image restoration [109], fluorescence image super-resolution [110, 111], and axial under-sampling mitigation [112].

Among these successful applications of deep learning, U-Net based neural networks have been the mainstream models. The U-Net was first proposed for 2D electron microscopy image segmentation [63] and later extended to other biological image transformation tasks, including cell detection and quantification [113]. In the field of augmented microscopy, most deep learning models directly apply U-Net based neural networks by only changing the loss functions for training [104, 107, 109, 72, 105]. In general, the U-Net is an encoder-decoder framework of neural network architectures for image transformation. It consists of a down-sampling path to capture multi-scale and multi-resolution contextual information, and a corresponding up-sampling path to enable precise voxel-wise predictions. Recent studies have enhanced the U-Net by incorporating residual blocks [10, 74, 65, 66] and supporting 3D image transformation [67].

Despite the success of these U-Net based neural networks for augmented microscopy, we observe three intrinsic limitations caused by the fact that they implement the encoder-decoder path by stacking local operators like convolutions and transposed convolutions with small kernels. First, in local operators, the size of receptive field (RF) of an output unit, determined by the kernel size, is usually small and does not aggregate information from the entire input. While stacking these local operators increases the size of RF for the final output units [114], the size of RF is still fixed given a specific neural network architecture. Each output unit follows a local path through the network and only has access to the information within its RF on the input image. Given a large input image, the network has to go deeper with more down-sampling and up-sampling operators to ensure each

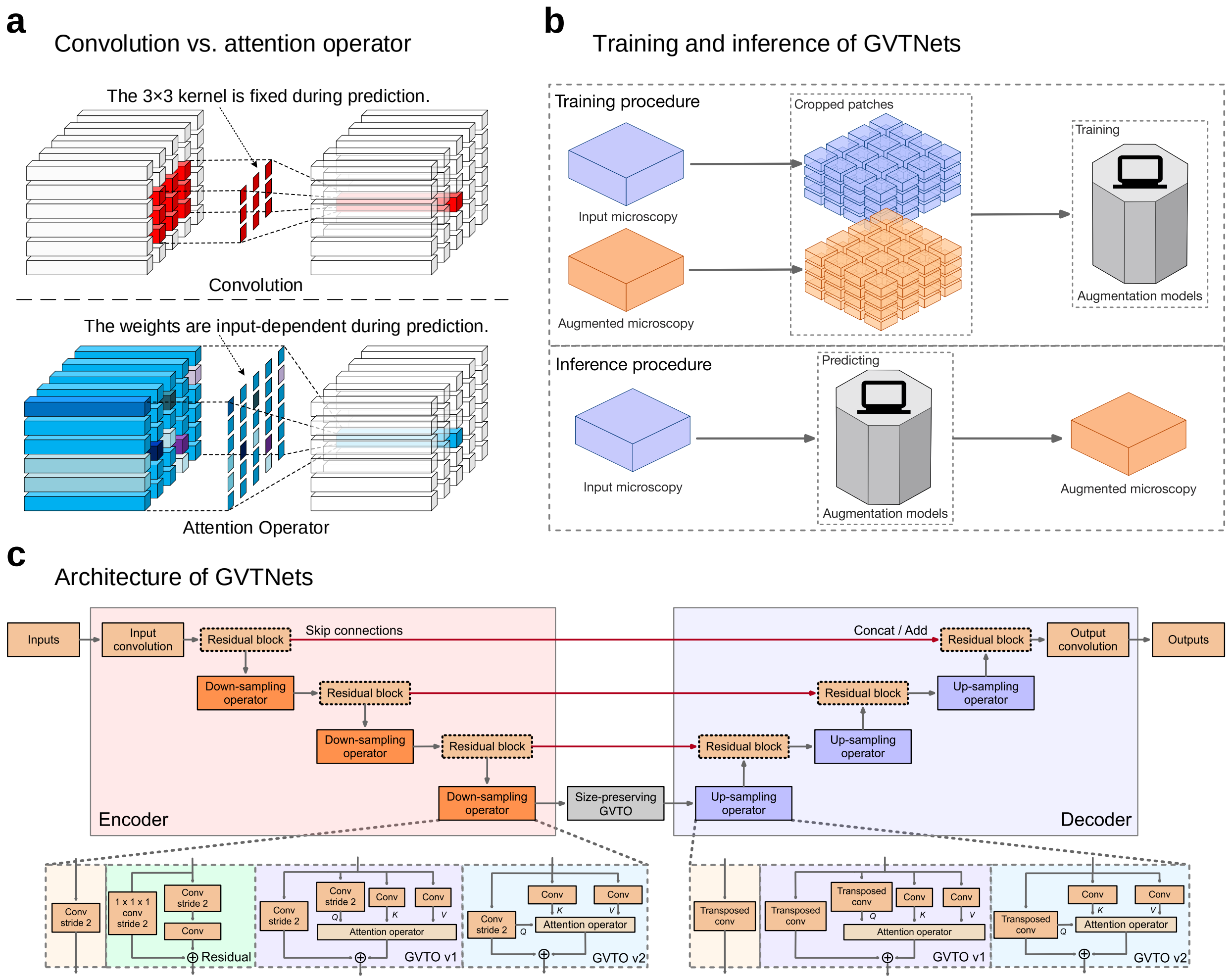


Figure 4.1: GVTNets architecture, training and inference. **a**, Comparisons between a 3×3 convolution and the attention operator in terms of receptive field and working mechanism during the inference procedure. The convolution, a typical local operator, has a fixed-size receptive field and fixed weights after training. On the contrary, the attention operator always allows a global receptive field and input-dependent weights during prediction. GVTOs, the key components of GVTNets, are built upon the attention operator. **b**, During the training procedure, registered pairs of microscope image before and after augmentation are collected and cut into small patches. During the inference procedure, the entire image is fed into the model to obtain the augmented output. **c**, A GVTNet of depth 4. The use of GVTOs differs GVTNet from the U-Net. The GVTNet fixes one size-preserving GVTO at the bottom level and allows optional GVTOs as down-sampling and up-sampling operators. The detailed description of GVTNets and GVTOs are provided in Section 4.3.

output unit received information from the entire input image. Such an approach is not efficient in terms of the amount of training parameters and computational expenses. In addition, the local path tends to focus on local dependencies among units and fails to capture long-range dependencies [13, 70], which are crucial for accuracy and consistency in biological image transformation. Second, the fixed-size RF limits the model's inference performance as well. The U-Net is usually trained with small patches of paired images, where cutting large images into small patches increases the amount of training data and stabilizes the training process by allowing large batch sizes [115]. As the U-Net produces the output of the same spatial size as the input, it is common to feed in the entire image or patches of much larger spatial sizes than the training patches during the prediction procedure, in order to speed up the inference [63, 107, 109, 113]. However, with the fixed-size RF, the model fails to take advantage of the knowledge from the entire input if the spatial size of the input is larger than that of RF, preventing potential inference performance boost. Third, all the local operators work with kernels whose weights are fixed after the training process, which means the importance of an input unit to an output unit is determined and not input-dependent during the inference stage. This property is helpful in detecting and extracting local patterns [8]. However, the model is supposed to be able to selectively use or ignore extracted information when transforming different input images, raising the need of operators that support input-dependent weights.

In this chapter, we argue that all three limitations above can be addressed by introducing the attention operator [13] into U-Net based neural networks. In order to demonstrate this point, we compare the attention operator with a typical local operator, *i.e.*, convolution, as shown in Fig. 4.1a. There are essential differences between the convolution and the attention operator. On one hand, the convolution has a local RF determined by its kernel, where each output unit receives information from a local area of input units. Meanwhile, note that the kernel weights are fixed after training. In other words, the weights do not depend on inputs during the inference. On the other hand, the attention operator computes each output unit as a weighted sum of all input units, where the weights are obtained through interactions between different representations of the inputs, as explained in Section 4.3. As a result, the attention operator is a non-local operator with a global receptive field, which can potentially overcome the first two limitations. In addition, the weights in the attention operator are input-dependent, addressing the third limitation.

Based on this insight, we build a family of non-local operators upon the attention operator, namely global voxel transformer operators (GVTOs). GVTOs organically combine local and non-local operators and can capture both local and long-range dependencies. In particular, GVTOs extend the attention operator to serve as flexible building blocks in the U-Net framework. Specifically, we develop GVTOs to support not only size-preserving, but also down-sampling and up-sampling tensor processing, which covers all kinds of operators in the U-Net framework. It is worth noting that, while GVTOs are designed for the U-Net framework, they can also be used in other kinds of networks as well.

With GVTOs, we propose global voxel transformer networks (GVTNets) as shown in Fig. 4.1c, an advanced deep learning tool for augmented microscopy, in order to address the limitations and improve current U-Net based neural networks. GVTNets follow the same encoder-decoder framework as the U-Net while using GVTOs instead of local operators only. To be concrete, we force GVTNets to connect the down-sampling and up-sampling paths using the size-preserving GVTO at the bottom level, which separates GVTNets from the U-Net. In addition, we allow users to flexibly use more GVTOs to replace local operators in the U-Net framework.

4.2 Related Work

In the literature, there exist many other studies that attempt to improve the U-Net in various aspects [116, 117, 118, 119, 120]. Among them, some studies [116, 117, 120] explore a similar direction to our work, which is to allow the U-Net to capture long-range dependencies or global context information. They can be mainly divided into two categories. One is to add modules composed of dilated convolutions, like Zhang et al. [116] and CE-NET [120]. Dilated convolutions can expand the receptive field of convolutions to capture longer-range dependencies. However, they are still local operators in essence, sharing similar limitations. For example, they cannot collect global information when inputs become larger than the receptive field. The other category is to apply global pooling to extract global information and use it to facilitate local operators, such as RSGU-Net [117]. However, important spatial information is lost during global pooling, which potentially limits the performance. Different from these two categories, we extend the attention operator to achieve the goal.

Other studies [118, 119] improve the U-Net in orthogonal directions. Oktay et al. [118] propose to add the gate mechanism to the skip connections, filtering out irrelevant information. It is worth noting that the gate mechanism and the attention mechanism are essentially different in terms of computation, functionality, and flexibility. The gate mechanism performs spatially element-wise filtering so that there is no explicit communication between spatial locations. On the contrary, the attention mechanism aggregates information from all spatial locations (Methods). Moreover, the gate mechanism can only be used for size-preserving tensor processing, while the attention mechanism can be extended for down-sampling and up-sampling tensor processing by our GVTOs. Zhou et al. [119] propose a nested U-Net architecture by adding dense skip connections. The nested architecture facilitates the training and yields better inference performance.

In terms of augmenting images with deep learning methods, generative adversarial network (GAN) [121] is a promising choice [122, 72, 101, 123]. We point out that GAN based methods are orthogonal to our GVTNets in the sense that they can be used together. Note that GAN is composed of a generator and a discriminator. In GAN based image augmentation models, the generator is typically a U-Net [123], which we can improve with our GVTNets.

4.3 Global Voxel Transformer Networks

4.3.1 Network Architecture

4.3.1.1 General Framework

Global voxel transformer networks (GVTNets) follow the same encoder-decoder framework as the U-Net [63, 67, 113], which represents a family of deep neural networks for biological image transformations. An encoder takes the image to be transformed as the input and computes feature maps of gradually reduced spatial sizes, which encode multi-scale and multi-resolution information from the input image. Then a corresponding decoder uses these feature maps to produce the transformed image, during which feature maps of gradually increased spatial sizes are computed. GVTNets support both 2D and 3D biological image transformations. We use the 3D case to describe the architecture in detail.

In our GVTNets, the encoder starts with an initial $3 \times 3 \times 3$ convolution that transforms the input image into a chosen number of feature maps of the same spatial size, initializing the encoding. The encoding process is achieved by down-sampling operators interleaved with optional size-preserving operators. Each down-sampling operator halves the size along each spatial dimension of feature maps but doubles the channel dimension, *i.e.*, the number of feature maps. To be specific, given an $d \times h \times w \times c$ tensor representing c feature maps of the spatial size $d \times h \times w$ as inputs, a down-sampling operator will output an $d/2 \times h/2 \times w/2 \times 2c$ tensor. Feature maps of the same spatial size are considered at the same level. As a result, the number of levels, also known as the depth of the network, is determined by the number of down-sampling operators in the encoder.

Correspondingly, the decoder is composed of the same number of up-sampling operators interleaved with optional size-preserving operators. The decoding process computes feature maps of increased spatial sizes in a level-by-level fashion, where each up-sampling operator doubles the size along each spatial dimension of feature maps but halves the channel dimension, as opposed to down-sampling operators. Therefore, there is a one-to-one correspondence between down-sampling and up-sampling operators. The decoder ends with an output convolution that outputs transformed image of the same spatial size as the input image.

The encoder and decoder are connected at each level. The bottom level contains the outputs of the encoder, which are feature maps of the smallest size in the U-Net framework. These feature maps, after optional size-preserving operators, serve as inputs to the decoder. In upper levels, there exist skip connections between the encoder and decoder. Concretely, the input feature maps to each down-sampling operator are concatenated or added to the output feature maps of the corresponding up-sampling operator. The skip connections allow the decoder to take advantage of encoded multi-scale and multi-resolution information, which increases the capability of the framework and facilitates the training process [63, 66].

4.3.1.2 From U-Net to Global Voxel Transformer Networks

The major difference between our GVTNets and the original U-Net lies in the choices of the size-preserving, down-sampling, and up-sampling operators, as shown in Fig. 4.1c. GVTNets are equipped with global voxel transformer operators (GVTOs), which can be flexibly used for size-preserving, down-sampling, or up-sampling tensor processing. In particular, GVTNets fix the size-preserving operator at the bottom level to be the size-preserving GVTO, ensuring that global information is encoded and aggregated before going through the decoder. The other size-preserving operators are set to pre-activation residual blocks [74], consisting of two $3 \times 3 \times 3$ convolutions with the ReLU activation function [9]. Down-sampling and up-sampling GVTOs can be used as corresponding operators based on the datasets and tasks.

4.3.2 Global Voxel Transformer Operators

As described above, the key components of our GVTNets are global voxel transformer operators (GVTOs), which are able to selectively use long-range information among input units. We take the 3D case to illustrate the size-preserving GVTO first, followed by the down-sampling and up-sampling GVTOs. Fig. 4.2 illustrates all different versions of GVTOs.

4.3.2.1 Size-preserving GVTO

Given the input third-order tensor $\mathcal{X} \in \mathbb{R}^{d \times h \times w \times c}$ representing c feature maps of the spatial size $d \times h \times w$, the size-preserving GVTO performs three independent $1 \times 1 \times 1$ convolutions on $\mathcal{X}$ and obtains three tensors, namely the query ($\mathcal{Q}$), key ($\mathcal{K}$), and value ($\mathcal{V}$) tensor, where $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V} \in \mathbb{R}^{d \times h \times w \times c}$. Afterwards, $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V}$ are unfolded along the channel dimension [124] into matrices $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V} \in \mathbb{R}^{c \times dhw}$. These matrices go through the attention operator defined as

$$\boldsymbol{Y} = \boldsymbol{V} \cdot \textsc{Normalize}(\boldsymbol{K}^T \boldsymbol{Q}) \in \mathbb{R}^{c \times dhw},$$

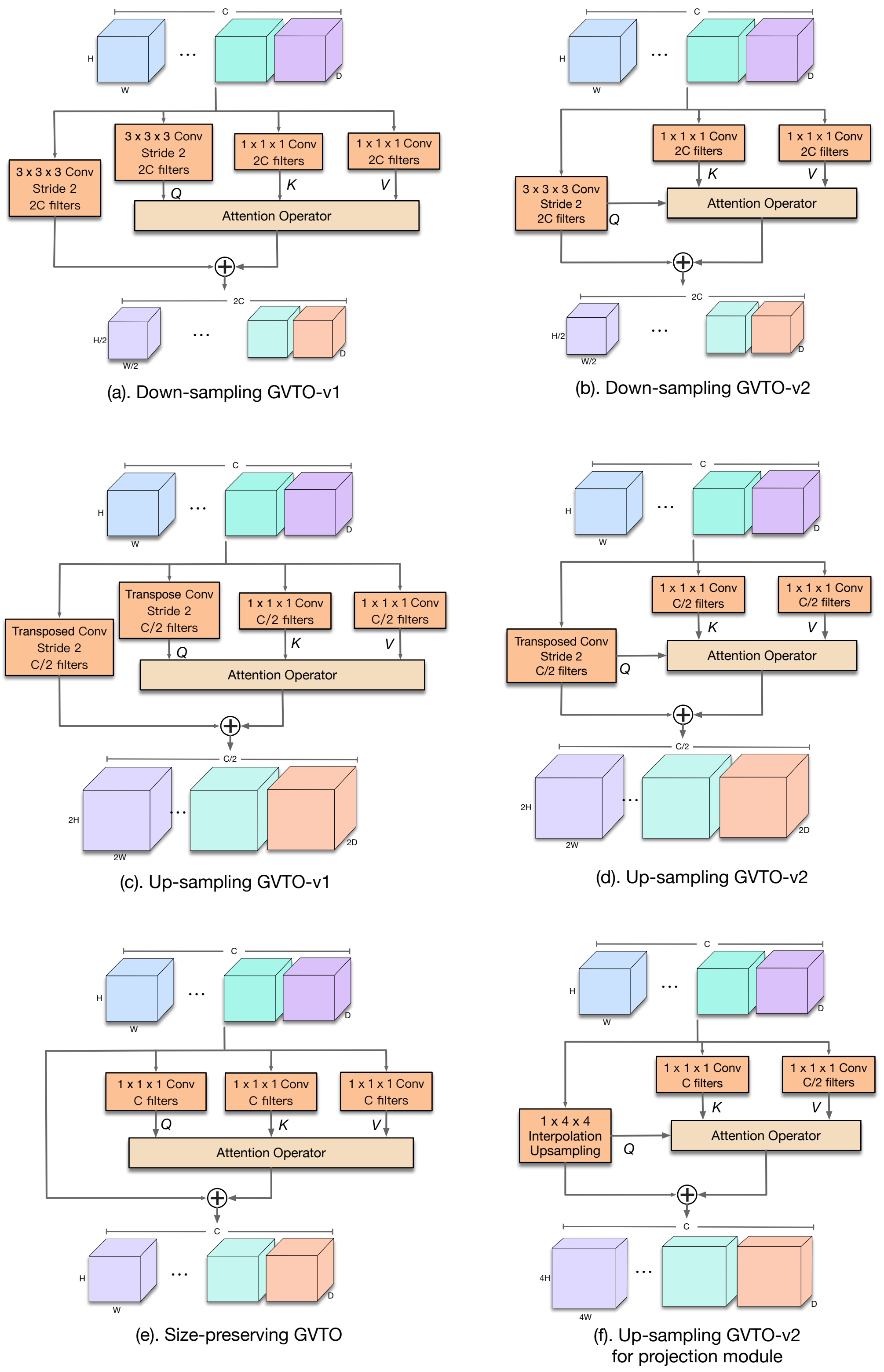

C
H
W
D
3 x 3 x 3 Conv Stride 2 2C filters
3 x 3 x 3 Conv Stride 2 2C filters
1 x 1 x 1 Conv 2C filters
1 x 1 x 1 Conv 2C filters
Q
K
V
Attention Operator
2C
H/2
W/2
(a). Down-sampling GVTO-v1
(b). Down-sampling GVTO-v2
Transposed Conv Stride 2 C/2 filters
Transpose Conv Stride 2 C/2 filters
1 x 1 x 1 Conv C/2 filters
C/2
2H
2W
2D
(c). Up-sampling GVTO-v1
(d). Up-sampling GVTO-v2
1 x 1 x 1 Conv C filters
(e). Size-preserving GVTO
1 x 4 x 4 Interpolation Upsampling
4H
4W
(f). Up-sampling GVTO-v2 for projection module

Figure 4.2: Global voxel transformer operators (GVTOs). **(a-b).** Two versions of down-sampling GVTOs that halve the spatial size but double the number of channels. **(c-d).** Two versions of up-sampling GVTOs that double the spatial size but halve the number of channels. **(e).** The size-preserving GVTO that keeps both the spatial size and the number of channels. **(f).** The special up-sampling GVTO used in the projection module for context-aware 3D to 2D image projection.

where $\textsc{Normalize}(\cdot)$ is a normalization function that normalizes each column of $\boldsymbol{Q}^T\boldsymbol{K} \in \mathbb{R}^{dhw \times dhw}$. Specifically, the size-preserving GVTO simply uses $1/dhw$ as the normalization function:

$$\boldsymbol{Y} = \boldsymbol{V}\frac{\boldsymbol{K}^T\boldsymbol{Q}}{dhw} = \frac{1}{dhw}\boldsymbol{V}\boldsymbol{K}^T\boldsymbol{Q} \in \mathbb{R}^{c \times dhw},$$

where dhw is the second dimension of $\boldsymbol{Q}$ and subjected to corresponding changes in the down-sampling and up-sampling GVTOs. After the attention operator, the matrix $\boldsymbol{Y}$ is then folded back to a tensor $\mathcal{Y} \in \mathbb{R}^{d \times h \times w \times c}$. The final outputs of the size-preserving GVTO is the summation of $\mathcal{X}$ and $\mathcal{Y}$, which means a residual connection from the inputs to the outputs [10]. In particular, we use the pre-activation technique as well [74]. As a result, the size-preserving GVTO preserves the dimension of the inputs.

4.3.2.2 Down-sampling and Up-sampling GVTOs

The extension from the size-preserving GVTO to the down-sampling and up-sampling GVTOs is achieved by changing the convolutions that compute $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V}$. We take the down-sampling GVTO as an example for illustration. Given the same input tensor $\mathcal{X} \in \mathbb{R}^{d \times h \times w \times c}$, we use a $3 \times 3 \times 3$ convolution with stride 2 to obtain $\mathcal{Q} \in \mathbb{R}^{d/2 \times h/2 \times w/2 \times 2c}$ and two independent $1 \times 1 \times 1$ convolutions to generate $\mathcal{K} \in \mathbb{R}^{d \times h \times w \times 2c}$ and $\mathcal{V} \in \mathbb{R}^{d \times h \times w \times 2c}$. The following computation is the same; that is, $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V}$ are unfolded along the channel dimension into matrices $\boldsymbol{Q} \in \mathbb{R}^{2c \times dhw/8}$ and $\boldsymbol{K}$, $\boldsymbol{V} \in \mathbb{R}^{2c \times dhw}$, which are fed into the same attention operator and output the matrix $\boldsymbol{Y} \in \mathbb{R}^{2c \times dhw/8}$. Folding it back results in a tensor $\mathcal{Y} \in \mathbb{R}^{d/2 \times h/2 \times w/2 \times 2c}$. Comparing the dimensions of $\mathcal{X}$ and $\mathcal{Y}$, we achieve a down-sampling process that halves the size along each spatial dimension of feature maps but doubles the channel dimension. We complete the down-sampling GVTO by adding the residual connection in two ways, corresponding to two versions of the down-sampling GVTO. One is to perform an extra $3 \times 3 \times 3$ convolution with stride 2 through the residual connection from $\mathcal{X}$ to $\mathcal{Y}$, in order to transform $\mathcal{X}$ to have the same dimension as $\mathcal{Y}$; the other is to directly add $\mathcal{Q}$ to $\mathcal{Y}$, based on the fact that $\mathcal{Q}$ is obtained from $\mathcal{X}$.

The up-sampling GVTO is dual to the down-sampling GVTO. Instead of using a convolution with stride 2, it uses a $3 \times 3 \times 3$ transposed convolution with stride 2 to obtain $\mathcal{Q} \in \mathbb{R}^{2d \times 2h \times 2w \times c/2}$. In addition, the other two $1 \times 1 \times 1$ convolutions generate $\mathcal{K} \in \mathbb{R}^{d \times h \times w \times c/2}$ and $\mathcal{V} \in \mathbb{R}^{d \times h \times w \times c/2}$. The up-sampling GVTO doubles the size along each spatial dimension of feature maps but halves the channel dimension and also has two versions corresponding to different residual connections.

4.3.2.3 Advantages of GVTOs

It is noteworthy that, each spatial location in the output tensor of GVTOs has access to all the information in the input tensor, and is able to selectively use or ignore information. We illustrate this point by regarding $\mathcal{X} \in \mathbb{R}^{d \times h \times w \times c}$ as $d \times h \times w$ c-dimensional vectors, where each vector represents the information in a spatial location. In this view, each vector has a one-to-one correspondence to each column in $\boldsymbol{K}$ and $\boldsymbol{V}$ in GVTOs, respectively. Revisiting the attention operator, each column in $\boldsymbol{Y}$ is a vector representation of each spatial location in the output tensor, and has a one-to-one correspondence to each column in $\boldsymbol{Q}$. Moreover, each column in $\boldsymbol{Y}$ is computed as the weighted sum of columns in $\boldsymbol{V}$, whose weights are determined by the interaction between the corresponding column in $\boldsymbol{Q}$ and all columns in $\boldsymbol{K}$. The weights can be viewed as filters of the amount of information from each spatial location in the inputs to the outputs. In addition, as both $\boldsymbol{Q}$ and $\boldsymbol{K}$ are computed from the input tensor, the weights are input-dependent. Therefore, GVTOs achieve the dynamic non-local information aggregation.

4.3.2.4 Comparisons with Fully-Connected Layers

It is important to note that the proposed GVTOs are different from fully-connected (FC) layers in fundamental ways, although they both allow each output unit to use information from the entire input. Compared to FC layers, outputs in GVTOs are computed based on relations among inputs. Thus the weights are input-dependent, rather than learned and fixed during prediction as in FC layers. The only trainable parameters in GVTOs are the convolutions to compute $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V}$, whose sizes are independent of input and output sizes. As a consequence, GVTOs allow variable-size inputs, and the positional correspondence between inputs and outputs is preserved in GVTOs. In contrast, FC layers require fixed-size inputs and positional correspondence is lost.

4.4 Experimental Studies

In the following, we (1) demonstrate the power of the basic GVTNets where only one size-preserving GVTO at the bottom level is applied, (2) show the effectiveness of employing more GVTOs in GVTNets and point out how GVTNets improve the inference performance, (3) explore the use of GVTOs in more complex and composite models, and (4) investigate the generalization ability of GVTNets.

4.4.1 Experimental Setup

All the experiments are conducted on publicly available datasets for augmented microscopy [107, 109, 101]. Details about datasets are introduced in each set of experiments in the following sections.

Global voxel transformer networks (GVTNets) are trained end-to-end under a supervised learning setting through back-propagation [8]. While the model aims at augmenting microscopy computationally, it still requires a relatively small amount of augmented microscopy images to be collected for training. Specifically, the training data are registered pairs of biological images before and after augmentation. Once trained, the model can be used to augment microscope images *in silico*, without involving any expensive microscopy hardware and technique. Following previous studies, we crop the training images into patches of smaller spatial sizes to train GVTNets. However, during the inference procedure, we feed in the entire image for prediction, as shown in Fig. 4.1b.

GVTNets are trained in an end-to-end fashion with two options of the loss functions. One is the mean squared error (MSE):

$$\mathcal{L}_{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where y represents the ground truth image, $\hat{y}$ represents the model's predicted image, and N represents the total number of voxels in the image. The other is the mean absolute error (MAE):

$$\mathcal{L}_{MAE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|.$$

Both MSE and MAE measure the differences between the predicted image and the ground truth image. The training process applies the Adam optimizer [84] with a user-chosen learning rate to minimize the loss.

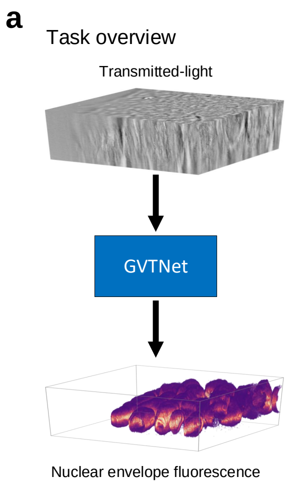
The settings of our device are - GPU: Nvidia GeForce RTX 2080 Ti 11GB; CPU: Intel Xeon Silver 4116 2.10GHz; OS: Ubuntu 16.04.3 LTS.

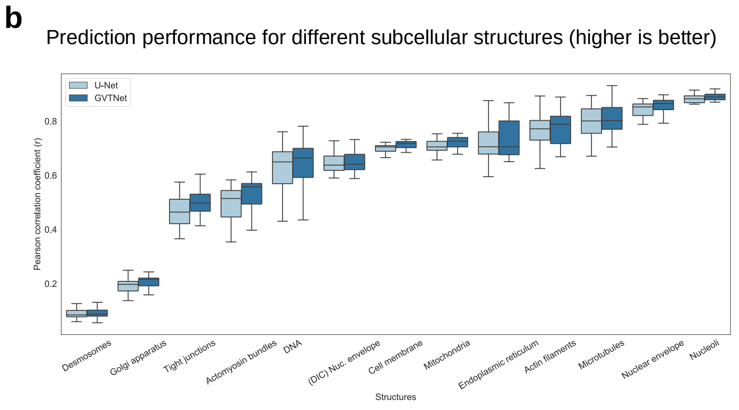
4.4.2 Label-free Prediction of 3D Fluorescence Images from Transmitted-light Microscopy

We first ask whether basic GVTNets achieve improved performance over U-Net based neural networks. A basic GVTNet differs from the U-Net only at the bottom level, by using a size-preserving GVTO instead of convolutions. The replacement is crucial, giving each output unit access to information from the entire input image, regardless of the spatial size. We apply a basic GVTNet on the public dataset from C. Ounkomol et al. [107], where the task is label-free prediction of 3D fluorescence images from transmitted-light microscopy, as illustrate in Fig. 4.3a.

The dataset is composed of 13 datasets corresponding to 13 different subcellular structures. All the images in the datasets are spatially registered and obtained from a database of images produced by the Allen Institute for Cell Science's microscopy pipeline [107]. The training and testing splits are provided by C. Ounkomol et al. [107] and available in our published code. For each structure, the training data are 30 spatially registered pairs of 3D transmitted-light images and ground truth fluorescence images. The number of testing images is 18 for the cell membrane, 10 for the differential interference contrast (DIC) nuclear envelope, and 20 for the others.

We use the model proposed by C. Ounkomol et al. [107] as the baseline model, which is the current state-of-the-art model on the 13 datasets. The baseline model is a U-Net based neural network of depth 5 containing $23,280,769$ training parameters, while the basic GVTNet that we used is of depth 4 containing $6,172,225$ training parameters. As a result, the basic GVTNet has





df	19	19	19	19	19	9	17	19	19	19	19	19	19
***P* value**	0.0418507	0.0039015	0.0013290	0.0007836	0.0017500	0.0124532	0.0000029	0.0000003	0.0268491	0.0308274	0.0045613	0.0000011	0.0002157

One-tailed paired t-tests

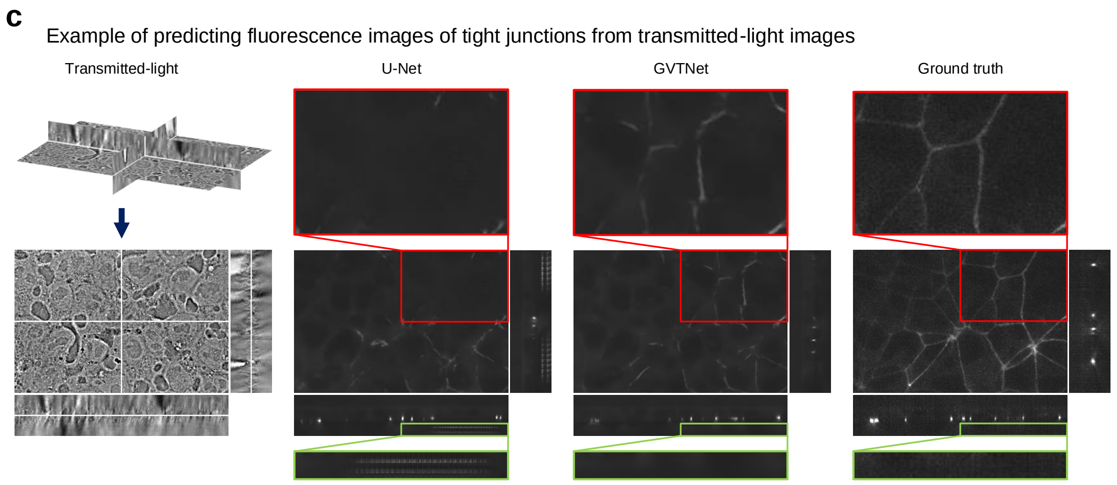

c
Example of predicting fluorescence images of tight junctions from transmitted-light images
Transmitted-light
U-Net
GVTNet
Ground truth

Figure 4.3: GVTNets on label-free prediction of 3D fluorescence images from transmitted-light microscopy. **a**, This augmented microscopy task is to predict the fluorescence images of subcellular structures from inexpensive transmitted-light images without fluorescent labels. **b**, Top: Distributions of the Pearson correlation coefficient (r) between the ground truth images and the predicted images of the GVTNet on the testing datasets for 13 different subcellular structures. Each pair of images leads to a point in the distribution. In the box and whisker plots, the 25th, 50th, and 75th percentile points are marked by the box, and whiskers indicate the minimum and maximum. The number of testing images is 18 for the cell membrane, 10 for the differential interference contrast (DIC) nuclear envelope, and 20 for the others. Bottom: One-tailed paired t-test results on the performance of the GVTNet and the U-Net baseline. The degree of freedom is the number of testing images minus one. All the *P* values are smaller than 0.05. **c**, From left to right, columns are transmitted-light input images, the predicted fluorescence images using the U-Net baseline, the predicted fluorescence images using the GVTNet, and the ground truth fluorescence images. We visualize the center z-, y-, and x-slices for 3D images. Clearly, the GVTNet captures more details. In addition, the GVTNet avoids artifacts caused by local operators.

Structure	Dataset	U-Net	GVTNet	P value
Nucleoli	fibrillarin	0.8759	**0.8857**	0.0002157
Nuclear envelope	lamin_b1	0.8432	**0.8567**	0.0000011
Microtubules	alpha_tubulin	0.7996	**0.8107**	0.0045613
Actin filaments	beta_actin	0.7579	**0.7673**	0.0308274
Endoplasmic reticulum	sec61_beta	0.7224	**0.7364**	0.0268491
Mitochondria	tom20	0.7033	**0.7187**	0.0000003
Cell membrane	membrane_caax_63x	0.6989	**0.7130**	0.0000029
(DIC) Nuc. envelope	dic_lamin_b1	0.6455	**0.6491**	0.0124532
DNA	dna	0.6259	**0.6411**	0.0017500
Actomyosin bundles	myosin_iib	0.4811	**0.5128**	0.0007836
Tight junctions	zo1	0.4598	**0.4994**	0.0013290
Glogi apparatus	st6gal1	0.1999	**0.2139**	0.0039015
Desmosomes	desmoplakin	0.0938	**0.0973**	0.0418507

Table 4.1: Comparisons of prediction performance between GVTNets and the U-Net for label-free prediction of 3D fluorescence images from transmitted-light microscopy. The comparisons are performed in terms of Pearson correlation coefficient (r) (higher is better). The statistics show the average r over the testing data. The number of testing images is 18 for the cell membrane, 10 for the differential interference contrast (DIC) nuclear envelope, and 20 for the others. The *P* values come from one-tailed paired t-tests, where the degree of freedom is the number of testing images minus one.

only 26.5% of training parameters of the baseline model. In addition, the computation speed becomes faster; that is, the GVTNet takes 0.4s to make prediction for one 3D image while the U-Net takes 1s [107].

The 13 subtasks corresponding 13 different subcellular structures are performed separately and independently. To train the GVTNet, the 30 pairs of training images are randomly cropped into patches of size $64 \times 64 \times 32$ and each training batch contains 16 pairs of patches. We minimize the MSE loss using the Adam optimizer with a learning rate of 0.001 for 70,000 to 100,000 minibatch iterations, depending on different subtasks. The training procedure lasts approximately 11h15m to 15h45m for each of the 13 datasets [107].

We quantify the model performance by computing the Pearson correlation coefficient on the testing data, define as

$$r(y, \hat{y}) = \frac{\sum_{i=1}^{N}(y_i - \mu_y)(\hat{y}_i - \mu_{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i - \mu_y)^2 \sum_{i=1}^{N}(\hat{y}_i - \mu_{\hat{y}})^2}},$$

where μ_y and $\mu_{\hat{y}}$ are the mean of voxel intensities in y and $\hat{y}$, respectively.

On all of the 13 datasets, our basic GVTNet consistently outperforms the U-Net baseline. We perform one-tailed paired t-tests and obtain *P* values smaller than 0.05 for all datasets, showing the improvements are statistically significant, as shown in Fig. 4.3b. The visualization of predictions in Fig. 4.3c indicates that the GVTNet captures more details than the U-Net baseline due to the access to more information, and is able to use global information to avoid local inconsistency. The quantitative testing results in terms of Pearson correlation coefficients are provided in Table 4.1. These experimental results indicate the effectiveness of only one size-preserving GVTO and the resulted basic GVTNets.

We note that both GVTNets and the U-Net baselines perform poorly on the datasets corresponding to Golgi apparatus and Desmosomes subcellular structures. According to C. Ounkomol et al. [107], a possible explanation is that the correlations between the input transmitted-light microscope images and the target fluorescence images are weak in these two datasets. As most supervised deep learning methods models try to capture the correlations between inputs and outputs during training, the inference performance could be poor if the correlations are weak.

4.4.3 Content-aware 3D Image Denoising

Next, we explore the potential of GVTNets by applying more GVTOs. Specifically, we apply GVTNets with both size-preserving and up-sampling GVTOs on two independent content-aware 3D image denoising tasks, as illustrated in Fig. 4.4a; namely, improving the signal-to-noise ratio (SNR) of live-cell imaging of *Planaria S. mediterranea* and developing *Tribolium castaneum* embryos.

The datasets were published by M. Weigert et al. [109], which contain pairs of 3D low-SNR images and ground truth high-SNR images for training and testing. The training data are provided

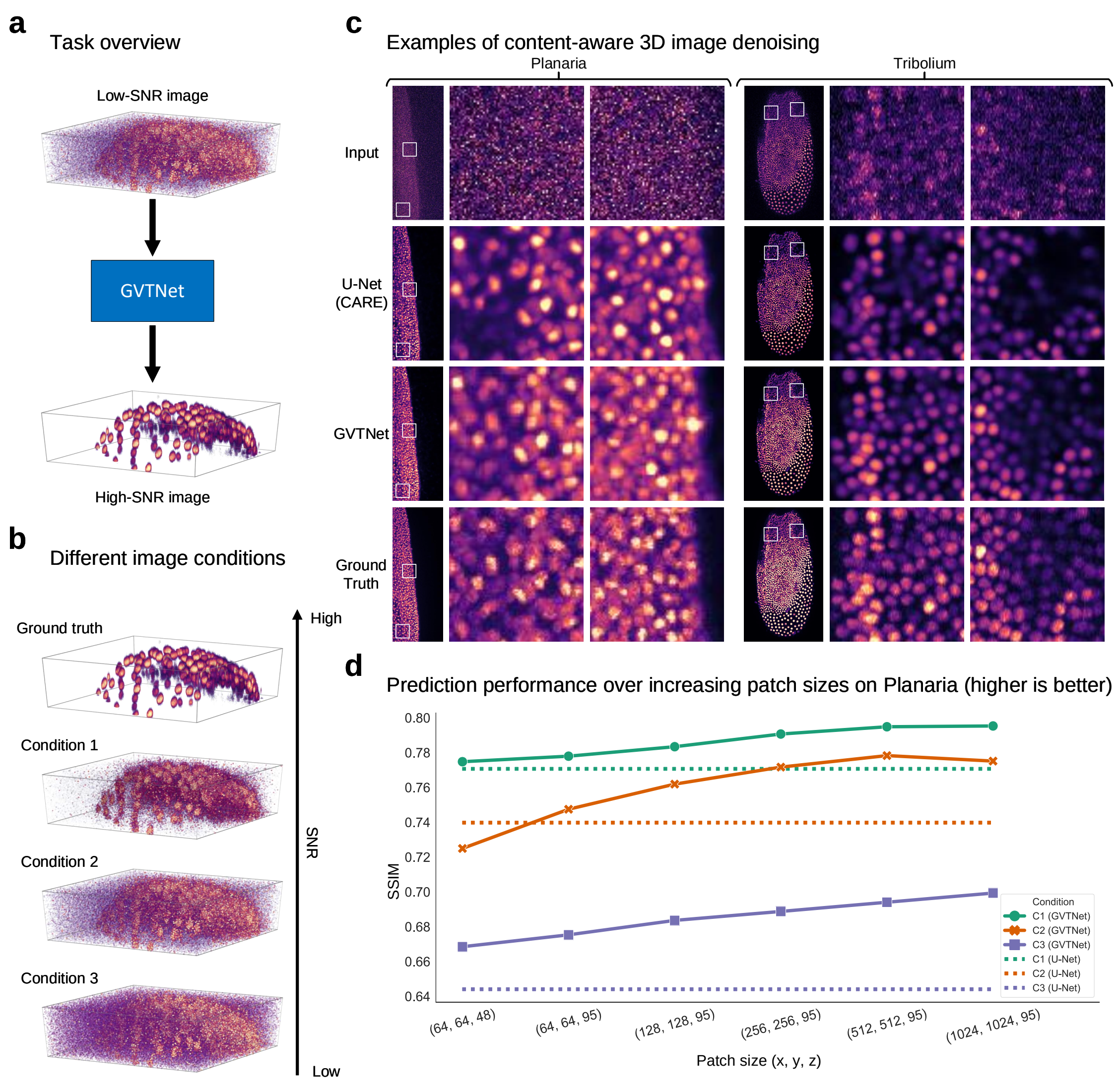


Figure 4.4: GVTNets on content-aware 3D image denoising. **a**, This augmented microscopy task is to improve the SNR by removing the noises from the low-SNR images captured in poor imaging conditions. **b**, The ground truth image captured with full exposure and lazer power condition, along with three noised images captured in weaker conditions at three different levels (C1-C3). **c**, From top to bottom, rows are the input noisy images, the predicted denoised images using the U-Net based CARE, the predicted denoised images using the GVTNet, and the ground truth denoised images. On both *Planaria* and *Tribolium* datasets, the U-Net fails to capture details in input regions with weak signals. On the contrary, the GVTNet obtains more precise predictions with more details in such regions. **d**, The inference performance in terms of SSIM over increasing prediction patch sizes on the *Planaria* dataset. The number of testing images is 20. Dotted lines represent the U-Net and solid lines represent the GVTNet. The inference performance of the GVTNet increases with larger prediction patch sizes, showing its ability of utilizing knowledge from the entire input.

in the form of 17,005 and 14,725 small cropped patches of size $64 \times 64 \times 16$ for *Planaria* and *Tribolium* datasets, while the testing data are 20 testing images of size $1024 \times 1024 \times 95$ and 6 testing images of average size around $700 \times 700 \times 45$ for the two datasets, respectively. In addition, the testing data come with three image conditions referring to three different SNR levels, leading to three degrees of denoising difficulty, as illustrated in Fig. 4.4b. Here, the image conditions refer to the laser-power and exposure-time during image collection [109]. Generally, low laser-power and short exposure-time lead to low SNR levels. Concretely, in the *Planaria* dataset, four different laser-power/exposure-time conditions are used: GT (ground truth) and C1–C3, specifically 2.31 mW/30 ms (GT), 0.12 mW/20 ms (C1), 0.12 mW/10 ms (C2), and 0.05 mW/10 ms (C3). Similarly, in the *Tribolium* dataset, four different laser-power imaging conditions are used: GT and C1–C3, specifically 20 mW (GT), 0.5 mW (C1), 0.2 mW (C2), and 0.1 mW (C3). As a result, each ground truth high-SNR image in testing dataset has three corresponding low-SNR images.

The baseline models in these experiments are the content-aware image restoration (CARE) networks [109], which are based on the 3D U-Net [67]. The U-Net based CARE networks achieve the current best performance on these two datasets, serving as a strong baseline. We build a GVTNet by replacing the bottom convolutions and up-sampling operators with corresponding size-preserving and up-sampling GVTOs. Specifically, our GVTNet follows a 3D U-Net framework of depth 3, i.e., including 2 down-sampling and up-sampling operators, respectively. The skip-connections merge feature maps from the encoder to the decoder by concatenation instead of addition. The bottom block is the size-preserving GVTO and two up-sampling operators are the up-sampling GVTOs v2. The number of feature maps after the initial convolution is set to 32. No batch normalization is applied.

We use the MAE loss with the Bayesian deep learning technique [125] to train the GVTNet. The training patch size is $64\times64\times16$. We train the model with a batch size of 16 and a base learning rate of 0.0004 with a decay rate 0.7 for every 10,000 minibatch iterations. The training procedure takes 50 epochs and lasts about 5h45m and 4h50m for the Planaria and Tribolium datasets [109], respectively.

		SSIM			NRMSE		
		Input	U-Net	GVTNet	Input	U-Net	GVTNet
Planaria	C1	0.2261	0.7707	**0.7929**	0.0777	0.0268	**0.0257**
	C2	0.1828	0.7397	**0.7745**	0.0822	0.0315	**0.0290**
	C3	0.1561	0.6441	**0.6972**	0.0851	0.0398	**0.0374**
Tribolium	C1	0.3677	0.9171	**0.9208**	0.0739	0.0241	**0.0234**
	C2	0.2356	0.9004	**0.9052**	0.0859	0.0282	**0.0271**
	C3	0.1823	0.8757	**0.8795**	0.0918	0.0340	**0.0336**

Table 4.2: **Comparisons of prediction performance between GVTNets and the U-Net for context-aware 3D image denoising.** The comparisons are performed in terms of the structural similarity index (SSIM) [126] (higher is better) and normalized root-mean-square error (NRMSE) (lower is better) on both *Planaria* and *Tribolium* datasets under three conditions (C1-C3). The baseline model is the U-Net based CARE [109]. The statistics show the average SSIM and NRMSE over the testing data. The number of testing images are 20 and 6 for *Planaria* and *Tribolium* datasets, respectively. The "Input" column shows the statistics computed between the input low-SNR images and the ground truth high-SNR images.

In order to quantify the model performance, we compute two evaluation metrics, *i.e.*, the structural similarity index (SSIM) [126] and normalized root-mean-square error (NRMSE), as defined below. The SSIM is computed as

$$SSIM(y, \hat{y}) = \frac{(2\mu_y \mu_{\hat{y}} + c_1)(2\sigma_{y\hat{y}} + c_2)}{(\mu_y^2 + \mu_{\hat{y}}^2 + c_1)(\sigma_y^2 + \sigma_{\hat{y}}^2 + c_2)},$$

where σ_y is the variance of y, $\sigma_{\hat{y}}$ is the variance of $\hat{y}$, $\sigma_{y\hat{y}}$ is the covariance of y and $\hat{y}$, and $c1 = (0.01L)^2$, $c2 = (0.03L)^2$ are two constant parameters of SSIM. Here, L represents the range of intensity values and is set to 1. The root-mean-square error (RMSE) is computed as

$$RMSE(y, \hat{y}) = \sqrt{\mathcal{L}_{MSE}(y, \hat{y})}.$$

Based on the RMSE, the NRMSE simply adds a normalization function on y and $\hat{y}$, respectively. In our tools and experiments, we apply the same percentile-based normalization and transformation as in M. Weigert et al. [109]. Concretely, the normalized root mean square error is defined by

$$NRMSE(y, \hat{y}) = \sqrt{\min_{\phi} \mathcal{L}_{MSE}(\phi(\hat{y}), \mathcal{N}(y, 0.1, 99.9))},$$

where

$$\mathcal{N}(y, 0.1, 99.9) = \frac{y - percentile(y, 0.1)}{percentile(y, 99.9) - percentile(y, 0.1)}$$

is the percentile-based normalization, and $\phi(\hat{y}) = \alpha\hat{y} + \beta$ denotes a transformation that scales and shifts $\hat{y}$. During the implementation, we let $\alpha = \frac{Cov(y-\bar{y}, \hat{y}-\bar{\hat{y}})}{Var(\hat{y}-\bar{\hat{y}})}$ and $\beta = 0$ to obtain $\phi(\hat{y})$ so that the MSE is minimized.

The models are evaluated under three SNR levels individually. The visualization results demonstrate that the GVTNet can take advantage of long-range dependencies to recover more details in areas with weak signals than the U-Net, as shown in Fig. 4.4c. The quantitative results also indicate significant and consistent improvements of the GVTNet over the U-Net based CARE under all image conditions on both datasets, revealing the advantages of GVTNets with more GVTOs, as reported in Table 4.2.

In order to provide insights on how GVTNets improve the inference performance by utilizing global information, we conduct extra experiments by varying the spatial sizes of input images during the inference process. To be specific, as both GVTNets and the U-Net are able to handle inputs of any spatial size, we can either feed the entire image directly into the model or crop the image into small prediction patches and reconstruct the entire augmented image after prediction. Theoretically, since the size of receptive filed (RF) in the U-Net is fixed and bounded, the prediction results will be the same as long as the size of prediction patches is larger than that of RF. On the other hand, the size of RF in GVTNets always cover the entire input image, allowing the use of more knowledge for better inference performance given large prediction patches. In order to verify this insight, we train the GVTNet and CARE on the *Planaria* dataset and compare prediction results in terms of SSIM when using prediction patches of sizes ranging from $64 \times 64 \times 48$ to $1024 \times 1024 \times 95$ (entire image size). The results are summarized in Fig. 4.4d. The prediction results of the U-Net remain the same when increasing prediction patch sizes, forming a horizontal line. On the contrary, significant improvements can be observed for the GVTNet. These results show that GVTNets are able to take advantage of larger prediction patches, which lead to a performance boost.

4.4.4 Content-aware 3D to 2D Image Projection

While we use GVTOs to build GVTNets, GVTOs are a family of operators that support any size-preserving, down-sampling and up-sampling tensor processing and can be used outside GVTNets. Therefore, we further examine the proposed GVTOs on more complicated and composite models. In particular, we apply GVTOs and GVTNets on the 3D *Drosophila melanogaster Flywing* surface projection task [127, 128], as illustrated in Fig. 4.5a.

The model for this task is supposed to take a noised 3D image as the input and projects it into a denoised 2D surface image. The typical deep learning model involves two parts; those are, a network for 3D to 2D surface projection, followed by a network for 2D image denoising. For example, the current best model, CARE [109], uses a task-specific convolutional neural network (CNN) [8] for projection and a 2D U-Net for denoising. The task-specific CNN is also composed of convolutions, down-sampling and up-sampling operators. We design our model based on CARE by applying GVTOs in the first CNN and replace the 2D U-Net with a 2D GVTNet. The resulted composite model employs size-preserving and up-sampling GVTOs in both parts.

During training, the 3D input patch size is $64 \times 64 \times 50$ and the 2D ground truth patch size is $64 \times 64 \times 1$. The other training settings are the same as those in image denoising experiments, except that we do not use the Bayesian deep learning technique. The training procedure lasts 4h55m for the Flywing dataset [109].

We compare our model with CARE on the *Flywing* dataset [109] in terms of SSIM and NRMSE. The dataset contains 16,891 pairs of small 3D noisy image patches and ground truth 2D surface image patches for training, and 26 complete images for testing.

The quantitative results indicate that the composite model augmented by GVTOs achieves significant improvements, as shown in Fig. 4.5c. We provide the detailed quantitative results in

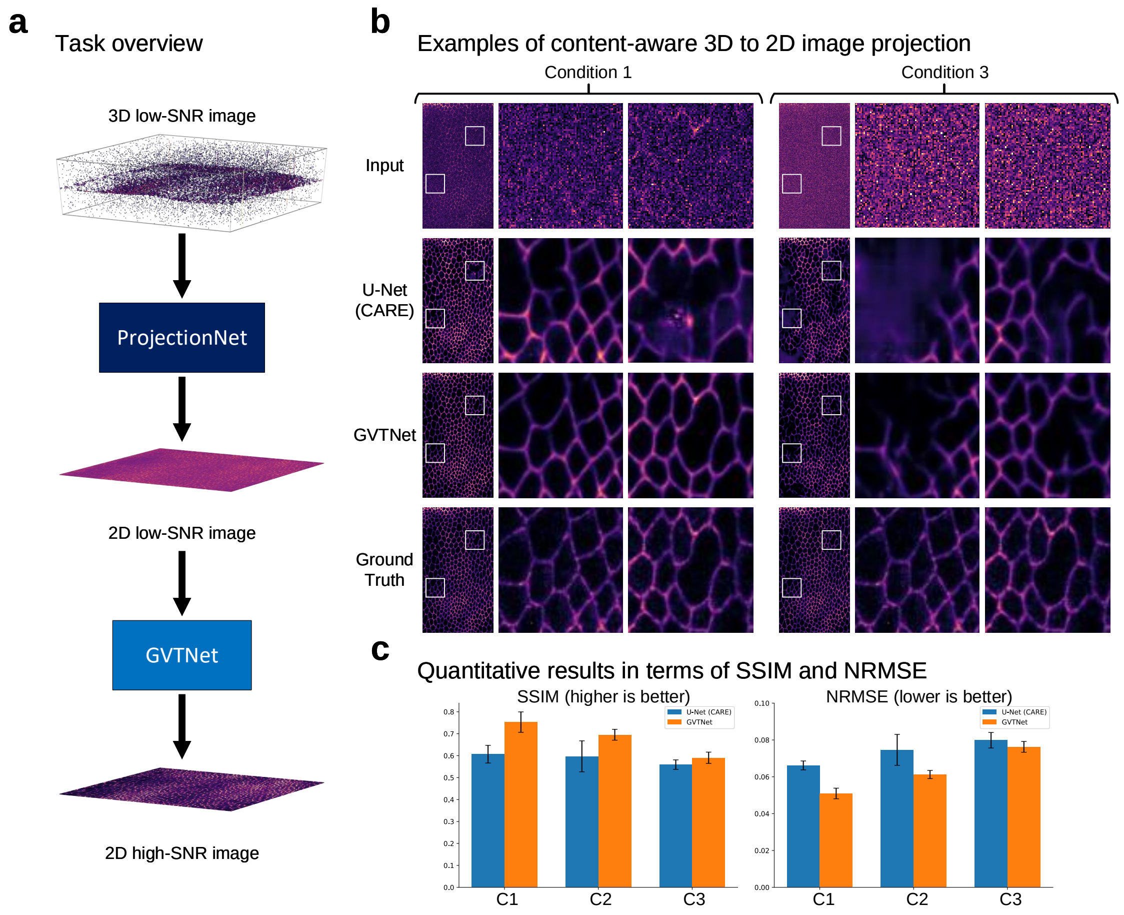


Figure 4.5: GVTNets on content-aware 3D to 2D image projection. **a**, This augmented microscopy task is to project a low-SNR 3D image into a high-SNR 2D surface. The model consists of a ProjectionNet that produces an intermediate 2D low-SNR image and a 2D GVTNet that outputs the high-SNR 2D image. **b**, From top to bottom, rows are input images, the predicted images using the U-Net based CARE, the predicted images using the GVTNet, and the ground truth images. Visualization results indicate that the U-Net is more sensitive to the irregular input voxel values and collapses in surrounding areas. In addition, the U-Net tends to give ambiguous and blurred predictions where the input information is insufficient. On the contrary, the GVTNet is more robust to these cases. **c**, Prediction performance on the testing data of the *Flywing* dataset, in terms of SSIM and NRMSE under three imaging conditions. The number of testing images is 26. The 68% confidence intervals are marked by computing the standard deviation over testing images.

Table 4.3. The visualization results show that the GVTOs have a stronger capability to recognize non-noisy objects at regions of lower SNR within an image, where the original model tends to fail, as shown in Fig. 4.5b. This is because the global information is of great importance to the projection tasks, especially along the Z-axis, where the projection happens. Specifically, for each

		SSIM			NRMSE		
		Input	U-Net	GVTNet	Input	U-Net	GVTNet
Flywing	C1	0.1902	0.6067	**0.7511**	0.1205	0.0662	**0.0513**
	C2	0.0795	0.5971	**0.6955**	0.1375	0.0746	**0.0612**
	C3	0.0241	0.5593	**0.5908**	0.1476	0.0798	**0.0762**

Table 4.3: Comparisons of prediction performance between GVTNets and the U-Net for context-aware 3D to 2D image projection. The comparisons are performed in terms of the structural similarity index (SSIM) [126] (higher is better) and normalized root-mean-square error (NRMSE) (lower is better) on the *Flywing* dataset under three conditions (C1-C3). The baseline model is the U-Net based CARE [109]. The statistics show the average SSIM and NRMSE over the testing data. The number of testing images is 26. The "Input" column shows the statistics computed between the input low-SNR images processed by PreMosa [129] and the ground truth high-SNR images.

(x, y) location in the 3D image, only one voxel along the Z-axis will be projected to the 2D surface. This restriction is only available when the model has the global information along the Z-axis. Therefore, plugging GVTOs into the projection process can effectively improve the overall performance.

4.4.5 Transfer Learning Ability of GVTNets

We have shown the effectiveness of GVTNets for augmented microscopy applications under a supervised learning setting. In the following, we further investigate the generalization ability of GVTNets under a simple transfer learning setting [130], where we train GVTNets on one dataset and perform testing on other datasets for the same task. In this case, the inconsistencies between the training and testing data often lead to the collapse of models based on local operators, such as the U-Net. One reasonable explanation is that the weights of kernels in local operators are fixed after training and independent to the inputs [130]. This limits the ability to deal with the different data distributions in training and inference procedures.

As GVTOs achieve input-dependent weights, we hypothesize that GVTNets are more robust to such inconsistencies and have a better generalization ability. We conduct experiments to verify the hypothesis using the three datasets from M. Weigert et al. [109]; namely, the *Planaria*, *Tribolium* and *Flywing* datasets. Note that all these datasets originally have 3D high-SNR ground truth images

Task	**Dataset**			**SSIM**		**NRMSE**	
	Training	Testing		U-Net	GVTNet	U-Net	GVTNet
Denoising	Planaria	Tribolium	C1	0.7346	**0.8726**	0.0366	**0.0324**
			C2	0.7909	**0.8439**	0.0404	**0.0385**
			C3	0.7368	**0.7615**	0.0487	**0.0478**
		Flywing	C1	0.3471	**0.3850**	0.0519	**0.0489**
			C2	0.2975	**0.3483**	0.0581	**0.0530**
			C3	0.2065	**0.2661**	0.0696	**0.0618**
Denoising	Tribolium	Planaria	C1	0.3093	**0.6374**	0.0675	**0.0455**
			C2	0.1986	**0.6060**	0.0773	**0.0486**
			C3	0.1810	**0.2928**	0.0840	**0.0710**
		Flywing	C1	0.0878	**0.0947**	0.1493	**0.0775**
			C2	0.0561	**0.0950**	0.1422	**0.0775**
			C3	0.0216	**0.0951**	0.1459	**0.0774**
Projection	Flywing	Planaria	C1	0.4275	**0.7264**	0.0675	**0.0640**
			C2	0.1846	**0.6774**	0.0773	**0.0716**
			C3	0.1675	**0.2648**	0.0840	**0.1171**
		Tribolium	C1	0.4341	**0.6504**	0.1487	**0.1181**
			C2	0.2743	**0.5905**	0.1704	**0.1364**
			C3	0.0453	**0.5018**	0.2058	**0.1617**

Table 4.4: Comparisons of transfer learning performance between GVTNets and the U-Net. The tasks are content-aware 3D image denoising for the first two experiments, and content-aware 3D to 2D image projection for the third experiment. The comparisons are performed in terms of the structural similarity index (SSIM) [126] (higher is better) and normalized root-mean-square error (NRMSE) (lower is better) under three conditions (C1-C3). The baseline model is the U-Net based CARE [109]. The statistics show the average SSIM and NRMSE over the testing data. The number of testing images are 20, 6, and 26 for *Planaria*, *Tribolium*, and *Flywing* datasets, respectively.

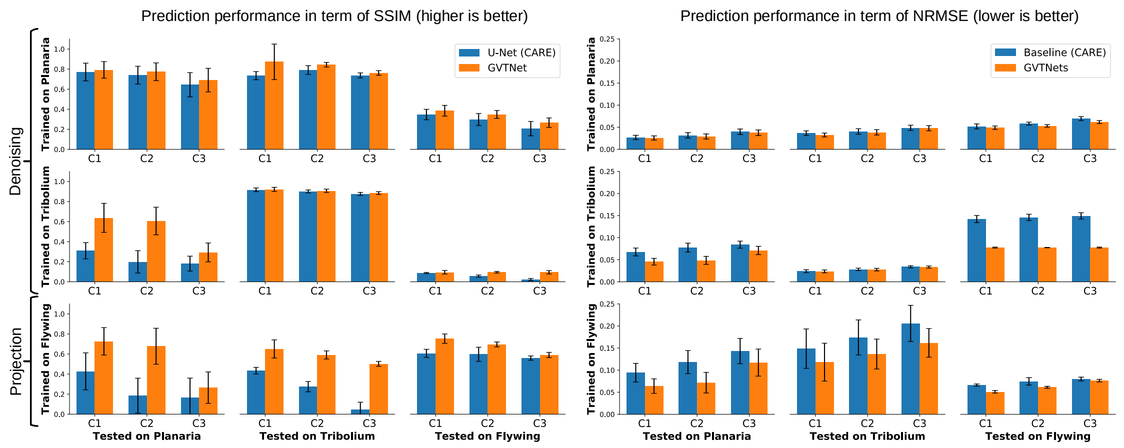


Figure 4.6: Generalization ability of GVTNets. Comparisons of the transfer learning performance between the U-Net base CARE and our GVTNets, in terms of SSIM (left) and NRMSE (right). Rows represent the dataset on which the models are trained and columns represent the dataset on which the models are tested. The first two rows correspond to the 3D denoising tasks and the third row corresponds to the 3D to 2D projection tasks. The diagonal charts are the performance of models trained and tested on the same datasets. The GVTNets can achieve a promising performance under this simplest transfer learning setting, due to the input-dependent weights of GVTOs. The 68% confidence intervals are marked by computing the standard deviation over testing images.

for the 3D denoising task. By applying PreMosa [129] on the 3D ground truth images, we can obtain 2D ground truth images for the 3D to 2D projection task. Therefore, these datasets can be used in either task for both training and testing. The baseline models are still the U-Net based CARE networks in these experiments, and we use the same GVTNet as introduced above for comparison. In general, we train GVTNet and CARE on one of the three datasets, and compare their testing performance on the remaining two datasets, resulting in three sets of experiments. To be concrete, the first two experiments where either the *Planaria* or *Tribolium* dataset is used for training are doing the 3D denoising tasks. The third experiment where models are trained on the *Flywing* dataset is performing the 3D to 2D projection task.

The comparison results in terms of SSIM and NRMSE are shown in Fig. 4.6. The detailed quantitative results can be found in Table 4.4. GVTNet obtains a more promising transfer learning performance than CARE, indicating a better generalization ability.

5. DEEP ATTENTION NETWORKS FOR GRAPHS: SECOND-ORDER POOLING FOR GRAPH NEURAL NETWORKS

In this chapter, we propose attentional second-order pooling as graph pooling, which is necessary for graph neural networks (GNNs) to perform graph classification tasks. In particular, we point out that the second-order pooling naturally satisfies the requirement of graph pooling but encounters practical problems. To overcome these problems, we bridge the second-order pooling with the attention mechanism and design an attention-based pooling method that can be flexibly used as either global or hierarchical graph pooling.[*]

5.1 Introduction

In recent years, deep learning has been widely explored on graph structured data, such as chemical compounds, protein structures, financial networks, and social networks [131, 132, 133]. Remarkable success has been achieved by generalizing deep neural networks from grid-like data to graphs [134, 135, 136, 137], resulting in the development of various graph neural networks (GNNs), like graph convolutional network (GCN) [15], GraphSAGE [16], graph attention network (GAT) [17], jumping knowledge network (JK) [138], and graph isomorphism networks (GINs) [3]. They are able to learn representations for each node in graphs and have set new performance records on tasks like node classification and link prediction [139]. In order to extend the success to graph representation learning, graph pooling is required, which takes node representations of a graph as inputs and outputs the corresponding graph representation.

While pooling is common in deep learning on grid-like data, it is challenging to develop graph pooling approaches due to the special properties of graphs. First, the number of nodes varies in different graphs, while the graph representations are usually required to have the same fixed size to fit into other machine learning models. Therefore, graph pooling should be capable of handling the variable number of node representations as inputs and producing fixed-sized graph representations.

[*]©2020 IEEE. Reprinted, with permission, from Zhengyang Wang and Shuiwang Ji, "Second-order pooling for graph neural networks.", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020

Second, unlike images and texts where we can order pixels and words according to the spatial structural information, there is no inherent ordering relationship among nodes in graphs. Indeed, we can set pseudo indices for nodes in a graph. However, an isomorphism of the graph may change the order of the indices. As isomorphic graphs should have the same graph representation, it is required for graph pooling to create the same output by taking node representations in any order as inputs.

Some previous studies employ simple methods such as averaging and summation as graph pooling [140, 141, 3]. However, averaging and summation ignore the feature correlation information, hampering the overall model performance [1]. Other studies have proposed advanced graph pooling methods, including DIFFPOOL [4], SORTPOOL [1], TOPKPOOL [142], SAGPOOL [6], and EIGENPOOL [5]. DIFFPOOL maps nodes to a pre-defined number of clusters but is hard to train. EIGENPOOL involves the computation of eigenvectors, which is slow and expensive. SORTPOOL, SAGPOOL and TOPKPOOL rely on the top-K sorting to select a fixed number (K) of nodes and order them, during which the information from unselected nodes is discarded. It is worth noting that all the existing graph pooling methods only collect first-order statistics [143].

In this chapter, we propose to use second-order pooling as graph pooling. Compared to existing graph pooling methods, second-order pooling naturally solves the challenges of graph pooling and is more powerful with its ability of using information from all nodes and collecting second-order statistics. We analyze the practical problems in directly using second-order pooling with GNNs. To address the problems, we propose two novel and effective global graph pooling approaches based on second-order pooling; namely, bilinear mapping and attentional second-order pooling. In addition, we extend attentional second-order pooling to hierarchical graph pooling for more flexible use in GNNs. We perform thorough experiments on ten graph classification benchmark datasets. The experimental results show that our methods improve the performance significantly and consistently.

5.2 Related Work

In this section, we review two categories of existing graph pooling methods in Section 5.2.1. Then in Section 5.2.2, we introduce what second-order statistics are, as well as their applications in both transitional machine learning and deep learning. In addition, we discuss the motivation of using second-order statistics in graph representation learning.

5.2.1 Graph Pooling: Global versus Hierarchical

Existing graph pooling methods can be divided into two categories according to their roles in graph neural networks (GNNs) for graph representation learning. One is global graph pooling, also known as graph readout operation [3, 6]. The other is hierarchical graph pooling, which is used to build hierarchical GNNs. We explain the details of the two categories and provide examples. In addition, we discuss advantages and disadvantages of the two categories.

Global graph pooling is typically used to connect embedded graphs outputted by GNN layers with classifiers for graph classification. Given a graph, GNN layers produce node representations, where each node is embedded as a vector. Global graph pooling is applied after GNN layers to process node representations into a single vector as the graph representation. A classifier takes the graph representation and performs graph classification. The "global" here refers to the fact that the output of global graph pooling encodes the entire graph. Global graph pooling is usually used only once in GNNs for graph representation learning. We call such GNNs as flat GNNs, in contrast to hierarchical GNNs. The most common global graph pooling methods include averaging and summation [140, 141, 3].

Hierarchical graph pooling is more similar to pooling in computer vision tasks [143]. The output of hierarchical graph pooling is a pseudo graph with fewer nodes than the input graph. It is used to build hierarchical GNNs, where hierarchical graph pooling is used several times between GNN layers to gradually decrease the number of nodes. The most representative hierarchical graph pooling methods are DIFFPOOL [4], SORTPOOL [1], TOPKPOOL [142], SAGPOOL [6], and EIGENPOOL [5]. A straightforward way to use hierarchical graph pooling for graph representation learning is to reduce the number of nodes to one. Then the resulted single vector is treated as the graph representation. Besides, there are two other ways to generate a single vector from the pseudo graph outputted by hierarchical graph pooling. One is introduced in SAGPOOL [6], where global and hierarchical graph pooling are combined. After each hierarchical graph pooling, global graph pooling with an independent classifier is employed. The final prediction is an average of all classifiers. On the other hand, SORTPOOL [1] directly applies convolutional neural networks (CNNs) to reduce the number of nodes to one. In particular, it takes advantage of a property of the pseudo graph outputted by hierarchical graph pooling. That is, the pseudo graph is a graph with a fixed number of nodes and there is an inherent ordering relationship among nodes determined by the trainable parameters in the hierarchical graph pooling. Therefore, common deep learning methods like convolutions can be directly used. In fact, we can simply concatenate node presentations following the inherent order as the graph representation.

Given this property, most hierarchical graph pooling methods can be flexibly used as global graph pooling, with the three ways introduced above. For example, SORTPOOL [1] is used to build flat GNNs and applied only once after all GNN layers. While the idea of learning hierarchical graph representations makes sense, hierarchical GNNs do not consistently outperform flat GNNs [6]. In addition, with advanced techniques like jumping knowledge networks (JK-Net) [138] to address the over-smoothing problem of GNN layers [144], flat GNNs can go deeper and achieve better performance than hierarchical GNNs [3].

In this chapter, we first focus on global graph pooling as second-order pooling naturally fits this category. Later, we extend one of our proposed graph pooling methods to hierarchical graph pooling in Section 5.3.6.

5.2.2 Second-order Statistics

In statistics, the k-order statistics refer to functions which use the k-th power of samples. Concretely, consider n samples $(x_1, x_2, \ldots, x_n)$. The first and second moments, *i.e.*, the mean $\mu = \frac{1}{n}\sum_i x_i$ and variance $\sigma^2 = \frac{1}{n}\sum_i (x_i - \mu)^2$, are examples of first and second-order statistics, respectively. If each sample is a vector, the covariance matrix is an example of second-order statistics. In terms of graph pooling, it is easy to see that existing methods are based on first-order statistics [143].

Second-order statistics have been widely explored in various computer vision tasks, such as face recognition, image segmentation, and object detection. In terms of traditional machine learning, the scale-invariant feature transform (SIFT) algorithm [145] utilizes second-order statistics of pixel values to describe local features in images and has become one of the most popular image descriptors. Tuzel et‘ al.[146, 147] use covariance matrices of low-level features with boosting for detection and classification. The Fisher encoding [148] applies second-order statistics for recognition as well. Carreira et al. [149] employs second-order pooling for semantic segmentation. With the recent advances of deep learning, second-order pooling is also used in CNNs for fine-grained visual recognition [150] and visual question answering [151, 152, 25].

Many studies motivates the use of second-order statistics as taking advantage of the Riemannian geometry of the space of symmetric positive definite matrices [153, 147, 149]. In these studies, certain regularizations are cast to guarantee that the applied second-order statistics are symmetric positive definite [154, 155]. Other work relates second-order statistics to orderless texture descriptors for images [148, 150].

In this chapter, we propose to incorporate second-order statistics in graph representation learning. Our motivations lie in three aspects. First, second-order pooling naturally fits the goal and requirements of graph pooling, as discussed in Sections 5.3.1 and 5.3.2. Second, second-order pooling is able to capture the correlations among features, as well as topology information in graph representation learning, as demonstrated in Section 5.3.2. Third, our proposed graph pooling methods based on second-order pooling are related to covariance pooling [146, 147, 154, 155] and attentional pooling [156] used in computer vision tasks, as pointed out in Section 5.3.5. In addition, we show that both covariance pooling and attentional pooling have certain limitations when employed in graph representation learning, and our proposed methods appropriately address them.

5.3 Second-order Pooling for Graphs

In this section, we introduce our proposed second-order pooling methods for graph representation learning. First, we formally define the aim and requirements of graph pooling in Section 5.3.1. Then we propose to use second-order pooling as graph pooling, analyze its advantages, and point out practical problems when directly using it with GNNs in Section 5.3.2. In order to address the problems, we propose two novel second-order pooling methods for graphs in Sections 5.3.3 and 5.3.4, respectively. Afterwards, we discuss why our proposed methods are more suitable as graph pooling compared to two similar pooling methods in image tasks in Section 5.3.5. Finally, while both methods focus on global graph pooling, we extend second-order pooling to hierarchical graph pooling in Section 5.3.6.

5.3.1 Properties of Graph Pooling

Consider a graph $G = (A, X)$ represented by its adjacency matrix $A \in \{0,1\}^{n\times n}$ and node feature matrix $X \in \mathbb{R}^{n\times d}$, where n is the number of nodes in G and d is the dimension of node features. The node features may come from node labels or node degrees. Graph neural networks (GNNs) are known to be powerful in learning good node representation matrix H from A and X:

$$H = [h_1, h_2, \ldots, h_n]^T = \text{GNN}(A, X) \in \mathbb{R}^{n\times f}, \tag{5.1}$$

where rows of H, $h_i \in \mathbb{R}^f, i = 1, 2, \ldots, n$, are representations of n nodes, and f depends on the architecture of GNNs. The task that we focus on in this chapter is to obtain a graph representation vector h_G from H, which is then fed into a classifier to perform graph classification:

$$h_G = g([A], H) \in \mathbb{R}^c, \tag{5.2}$$

where $g(\cdot)$ is the graph pooling function and c is the dimension of h_G. Here, $[A]$ means that the information from A can be optionally used in graph pooling. For simplicity, we omit it in the following discussion.

Note that $g(\cdot)$ must satisfy two requirements to serve as graph pooling. First, $g(\cdot)$ should be able to take H with variable number of rows as the inputs and produce fixed-sized outputs. Specifically, different graphs may have different number of nodes, which means that n is a variable. On the other hand, c is supposed to be fixed to fit into the following classifier.

Second, $g(\cdot)$ should output the same h_G when the order of rows of H changes. This permutation invariance property is necessary to handle isomorphic graphs. To be concrete, if two graph $G_1 = (A_1, X_1)$ and $G_2 = (A_2, X_2)$ are isomorphic, GNNs will output the same multiset of node representations [1, 3]. That is, there exists a permutation matrix $P \in \{0,1\}^{n\times n}$ such that $H_1 = PH_2$, for $H_1 = \text{GNN}(A_1, X_1)$ and $H_2 = \text{GNN}(A_2, X_2)$. However, the graph representation computed by $g(\cdot)$ should be the same, *i.e.*, $g(H_1) = g(H_2)$ if $H_1 = PH_2$.

5.3.2 Second-order Pooling

In this chapter, we propose to employ second-order pooling [149], also known as bilinear pooling [150], as graph pooling. We show that second-order pooling naturally satisfies the two requirements above.

We start by introducing the definition of second-order pooling.

Definition. Given $H = [h_1, h_2, \ldots, h_n]^T \in \mathbb{R}^{n\times f}$, second-order pooling (SOPOOL) is defined as

$$\text{SOPOOL}(H) = \sum_{i=1}^{n} h_i h_i^T = H^T H \in \mathbb{R}^{f\times f}. \tag{5.3}$$

In terms of graph pooling, we can view $\text{SOPOOL}(H)$ as an f^2-dimensional graph representation vector by simply flattening the matrix. Another way to transform the matrix into a vector is discussed in Section 5.3.4. Note that, as long as SOPOOL meets the two requirements, the way to transform the matrix into a vector does not affect its eligibility as graph pooling.

Now let us check the two requirements.

Proposition 1. SOPOOL always outputs an $f \times f$ matrix for $H \in \mathbb{R}^{n\times f}$, regardless of the value of n.

Proof. The result is obvious since the dimension of $H^T H$ does not depend on n. □

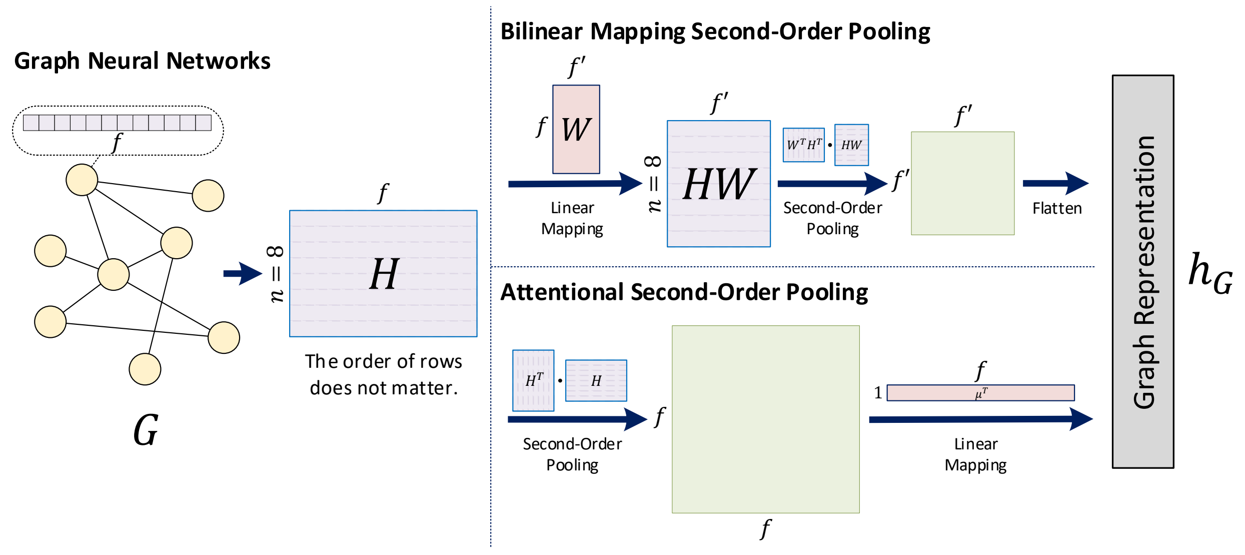


Figure 5.1: Illustrations of our proposed graph pooling methods: bilinear mapping second-order pooling (SOPOOL$_{bimap}$) in Section 5.3.3 and attentional second-order pooling (SOPOOL$_{attn}$) in Section 5.3.4. This is an example for a graph G with $n = 8$ nodes. GNNs can learn representations for each node and graph pooling processes node representations into a graph representation vector h_G. ©2020 IEEE

Proposition 2. SOPOOL is invariant to permutation so that it outputs the same matrix when the order of rows of H changes.

Proof. Consider $H_1 = PH_2$, where P is a permutation matrix. Note that we have $P^TP = I$ for any permutation matrix. Therefore, it is easy to derive

$$\begin{aligned} \text{SOPOOL}(H_1) &= H_1^T H_1 \\ &= (PH_2)^T(PH_2) \\ &= H_2^T P^T P H_2 \\ &= H_2^T H_2 = \text{SOPOOL}(H_2). \end{aligned} \tag{5.4}$$

This completes the proof. □

In addition to satisfying the requirements of graph pooling, SOPOOL is capable of capturing second-order statistics, which are much more discriminative than first-order statistics computed by most other graph pooling methods [149, 150, 151]. In detail, the advantages can be seen from two aspects. On one hand, we can tell from $\text{SOP}\textsc{ool}(H) = \sum_{i=1}^{n} h_i h_i^T$ that, for each node representation h_i, the features interact with each other, enabling the correlations among features to be captured. On the other hand, topology information is encoded as well. Specifically, we view $H \in \mathbb{R}^{n \times f}$ as $H = [l_1, l_2, \ldots, l_f]$, where $l_j \in \mathbb{R}^n$, $j = 1, 2, \ldots, f$. The vector l_j encodes the spatial distribution of the j-th feature in the graph. Based on this view, $\text{SOP}\textsc{ool}(H) = H^T H$ is able to capture the topology information.

However, we point out that the direct application of second-order pooling in GNNs leads to practical problems. The direct way to use second-order pooling as graph pooling is represented as

$$h_G = \text{F}\textsc{latten}(\text{SOP}\textsc{ool}(\text{GNN}(A, X))) \in \mathbb{R}^{f^2}. \tag{5.5}$$

That is, we apply $\text{SOP}\textsc{ool}$ on $H = \text{GNN}(A, X)$ and flatten the output matrix into an f^2-dimensional graph representation vector. However, it causes an explosion in the number of training parameters in the following classifier when f is large, making the learning process harder to converge and easier to overfit. While each layer in a GNN usually has outputs with a small number of hidden units (e.g. 16, 32, 64), it has been pointed out that graph representation learning benefits from using information from outputs of all layers, obtaining better performance and generalization ability [138]. It is usually achieved by concatenating outputs across all layers in a GNN [1, 3]. In this case, H has a large final f, making direct use of second-order pooling infeasible. For example, if a GNN has 5 layers and each layer's outputs have 32 hidden units, f becomes $32 \times 5 = 160$. Suppose h_G is sent into a 1-layer fully-connected classifier for c graph categories in a graph classification task. It results in $160^2 c = 25,600c$ training parameters, which is excessive. We omit the bias term for simplicity.

5.3.3 Bilinear Mapping Second-order Pooling

To address the above problem, a straightforward solution is to reduce f in H before $\text{SOP}\textsc{ool}(H)$. Based on this, our first proposed graph pooling method, called bilinear mapping second-order pooling ($\text{SOP}\textsc{ool}_{bimap}$), employs a linear mapping on H to perform dimensionality reduction. Specifically, it is defined as

$$\begin{aligned}\text{SOP}\textsc{ool}_{bimap}(H) &= \text{SOP}\textsc{ool}(HW) \\ &= W^T H^T H W \in \mathbb{R}^{f' \times f'}, \end{aligned} \tag{5.6}$$

where $f' < f$ and $W \in \mathbb{R}^{f \times f'}$ is a trainable matrix representing a linear mapping. Afterwards, we follow the same process to flatten the matrix and obtain an f'^2-dimensional graph representation vector:

$$h_G = \textsc{Flatten}(\text{SOP}\textsc{ool}_{bimap}(\text{GNN}(A, X))) \in \mathbb{R}^{f'^2}. \tag{5.7}$$

Figure 5.1 provides an illustration of the above process. By selecting an appropriate f', the bilinear mapping second-order pooling does not suffer from the excessive number of training parameters. Taking the example above, if we set $f' = 32$, the total number of parameters in $\text{SOP}\textsc{ool}_{bimap}$ and a following 1-layer fully-connected classifier is $32 \times 160 + 32^2 c = 5,120 + 1,024c$, which is much smaller than $25,600c$.

5.3.4 Attentional Second-order Pooling

Our second proposed graph pooling method tackles with the problem by exploring another way to transform the matrix computed by $\text{SOP}\textsc{ool}$ into the graph representation vector, instead of simply flattening. Similarly, we use a linear mapping to perform the transformation, defined as

$$h_G = \text{SOP}\textsc{ool}(\text{GNN}(A, X)) \cdot \mu \in \mathbb{R}^f, \tag{5.8}$$

where $\mu \in \mathbb{R}^f$ is a trainable vector. It is interesting to note that $h_G = H^T H \mu$, which is similar to the sentence attention in [26]. To be concrete, consider a word embedding matrix $E = [e_1, e_2, \ldots, e_l]^T \in \mathbb{R}^{l \times d_w}$ for a sentence, where l is the number of words and d_w is the dimension

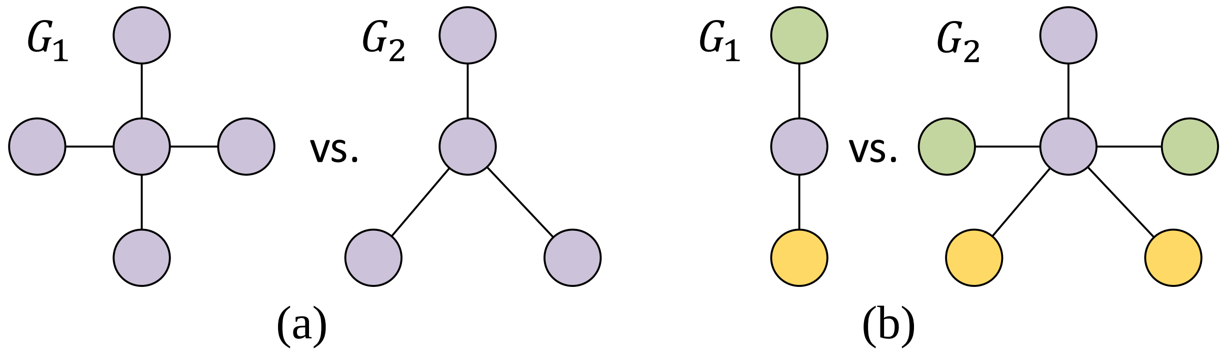


Figure 5.2: Examples of graphs that pooling methods discussed in Section 5.3.5 fail to distinguish, *i.e.*, producing the same graph representation for different graphs G_1 and G_2. The same color denotes the same node representation. (a) Covariance pooling (CovPool) and attentional pooling (AttnPool) both fail. CovPool fails because subtracting the mean results in $h_{G_1} = h_{G_2} = \mathbf{0}$. AttnPool computes the mean of node representations, leading to $h_{G_1} = h_{G_2}$ as well. (b) AttnPool fails in this example with the same μ. ©2020 IEEE

of word embeddings. The sentence attention is defined as

$$\alpha_i = \frac{\exp(e_i^T \mu_s)}{\sum_{j=1}^{l} \exp(e_j^T \mu_s)}, i = 1, 2, \ldots, l \tag{5.9}$$

$$s = \sum_{i=1}^{l} \alpha_i e_i, \tag{5.10}$$

where $\mu_s \in \mathbb{R}^{d_w}$ is a trainable vector and s is the resulted sentence embedding. Note that Eqn. (5.9) is the Softmax function and serves as a normalization function [13]. Rewriting the sentence attention into matrix form, we have $s = E^T \text{Softmax}(E\mu_s)$. The only difference between the computation of h_G and that of s is the normalization function. Therefore, we name our second proposed graph pooling method as attentional second-order pooling (SOPool$_{attn}$), defined as

$$\text{SOPool}_{attn}(H) = H^T H \mu \in \mathbb{R}^f, \tag{5.11}$$

where $\mu \in \mathbb{R}^f$ is a trainable vector. It is illustrated in Figure 5.1. We take the same example above to show that SOPool$_{attn}$ reduces the number of training parameters. The total number of parameters in SOPool$_{attn}$ and a following 1-layer fully-connected classifier is just $160 + 160c$, significantly reducing the amount of parameters compared to $25,600c$.

5.3.5 Relationships to Covariance Pooling and Attentional Pooling

The experimental results in Section 5.4 show that both our proposed graph pooling methods achieve better performance significantly and consistently than previous studies. However, we note that, there are pooling methods in image tasks that have similar computation processes to our proposed methods, although they have not been developed based on second-order pooling. In this section, we point out the key differences between these methods and ours and show why they matter in graph representation learning.

Note that images are usually processed by deep neural networks into feature maps $I \in \mathbb{R}^{h \times w \times c}$, where h, w, c are the height, width, and number of feature maps, respectively. Following [156, 154, 155], we reshape I into the matrix $H \in \mathbb{R}^{n \times f}$, where $n = hw$ and $f = c$ so that different pooling methods can be compared directly.

Covariance pooling. Covariance pooling (CovPool) [146, 147, 154, 155] has been widely explored in image tasks, such as image categorization, facial expression recognition, and texture classification. Recently, it has also been explored in GNNs [2]. The definition is

$$\textsc{CovPool}(H) = (H - \mathbf{1}\bar{H})^T(H - \mathbf{1}\bar{H}) \in \mathbb{R}^{f \times f}, \tag{5.12}$$

where $\mathbf{1}$ is the n-dimensional all-one vector and $\bar{H} \in \mathbb{R}^{1 \times f}$ is the mean of rows of H. It differs from SOPool defined in Eqn. (5.3) only in whether to subtract the mean. However, subtracting the mean makes CovPool less powerful in terms of distinguishing graphs with repeating node embeddings [3], which may cause the performance loss. Figure 5.2(a) gives an example of this problem.

Attentional pooling. Attentional pooling (AttnPool) [156] has been used in action recognition. As shown in Section 5.3.4, it is also used in text classification [26], defined as

$$\textsc{AttnPool}(H) = H^T \textsc{Softmax}(H\mu) \in \mathbb{R}^{f}, \tag{5.13}$$

where $\mu \in \mathbb{R}^f$ is a trainable vector. It differs from SOPOOL$_{attn}$ only in the SOFTMAX function. We show that the SOFTMAX function leads to similar problems as other normalization functions, such as mean and max-pooling [3]. Figure 5.2 provides examples in which ATTNPOOL does not work.

To conclude, our methods derived from second-order pooling are more suitable as graph pooling. We compare these pooling methods through experiments in Section 5.4.3. The results show that COVPOOL and ATTNPOOL suffer from significant performance loss on some datasets.

5.3.6 Multi-head Attentional Second-order Pooling

The proposed SOPOOL$_{bimap}$ and SOPOOL$_{attn}$ belong to the global graph pooling category. As discussed in Section 5.2.1, they are used in flat GNNs after all GNN layers and output the graph representation for the classifier. While flat GNNs outperform hierarchical GNNs in most benchmark datasets [3], developing hierarchical graph pooling is still desired, especially for large graphs [4, 5, 6]. Therefore, we explore a hierarchical graph pooling method based on second-order pooling.

Unlike global graph pooling, hierarchical graph pooling outputs multiple vectors corresponding to node representations in the pooled graph. In addition, hierarchical graph pooling has to update the adjacency matrix to indicate how nodes are connected in the pooled graph. To be specific, given the adjacency matrix $A \in \mathbb{R}^{n \times n}$ and node representation matrix $H \in \mathbb{R}^{n \times f}$, a hierarchical graph pooling function $g_h(\cdot)$ can be written as

$$A', H' = g_h(A, H), \tag{5.14}$$

where $A' \in \mathbb{R}^{k \times k}$ and $H' \in \mathbb{R}^{k \times f}$. Here, k is a hyperparameter determining the number of nodes in the pooled graph. Note that Eqn. (5.14) does not conflict with Eqn. (5.2), as we can always transform H' into a vector h_G, as discussed in Section 5.2.1.

We note that the proposed SOPOOL$_{attn}$ in Section 5.3.4 is closely related to the attention mechanism and can be easily extended to a hierarchical graph pooling method based on the multi-head technique in the attention mechanism [13, 17]. The multi-head technique means that multiple independent attentions are performed on the same inputs. Then the outputs of multiple attentions are then concatenated together. Based on this insight, we propose multi-head attentional second-order pooling ($\text{SOP}\textsc{ool}_{m\_attn}$), defined as

$$H' = \text{SOP}\textsc{ool}_{m\_attn}(H) = UH^TH \in \mathbb{R}^{k\times f}, \tag{5.15}$$

where $U \in \mathbb{R}^{k\times f}$ is a trainable matrix. To illustrate its relationship to the multi-head technique, we can equivalently write it as

$$\text{SOP}\textsc{ool}_{m\_attn}(H) = [H^TH\mu_1, \ldots, H^TH\mu_k]^T, \tag{5.16}$$

where we decompose U in Eqn. (5.15) as $U = [\mu_1, \mu_2, \ldots, \mu_k]^T$. The relationship can be easily seen by comparing Eqn. (5.16) with Eqn. (5.11).

The multi-head technique enables $\text{SOP}\textsc{ool}_{m\_attn}$ to output the node representation matrix for the pooled graph. We now describe how to update the adjacency matrix. In particular, we employ a contribution matrix C in updating the adjacency matrix. The contribution matrix is a $k \times n$ matrix, whose entries indicate how nodes in the input graph contribute to nodes in the pooled graph. In $\text{SOP}\textsc{ool}_{m\_attn}$, we can simply let $C = UH^T \in \mathbb{R}^{k\times n}$. With the contribution matrix C, the corresponding adjacency matrix A' of the pooled graph can be computed as

$$A' = CAC^T \in \mathbb{R}^{k\times k}. \tag{5.17}$$

The proposed $\text{SOP}\textsc{ool}_{m\_attn}$ is closely related to DIFFPOOL [4]. The contribution matrix C corresponds to the assignment matrix in DIFFPOOL. However, DIFFPOOL applied GNN layers with normalization on H to obtain C, preventing the explicit use of second-order statistics. In the experiments, we evaluate $\text{SOP}\textsc{ool}_{m\_attn}$ as both global and hierarchical graph pooling methods, in flat and hierarchical GNNs, respectively.

5.4 Experimental Studies

We conduct thorough experiments on graph classification tasks to show the effectiveness of our proposed graph pooling methods, namely bilinear mapping second-order pooling ($\text{SOP}\textsc{ool}_{bimap}$), attentional second-order pooling ($\text{SOP}\textsc{ool}_{attn}$), and multi-head attentional second-order pooling ($\text{SOP}\textsc{ool}_{m\_attn}$). Section 5.4.1 introduces the datasets, baselines, and experimental setups for reproduction. The following sections aim at evaluating our proposed methods in different aspects, by answering the questions below:

- Can GNNs with our proposed methods achieve improved performance in graph classification tasks? Section 5.4.2 provides the comparison results between our methods and existing methods in graph classification tasks.

- Do our proposed methods outperform existing global graph pooling methods with the same flat GNN architecture? The ablation studies in Section 5.4.3 compare different graph pooling methods with the same GNN, eliminating the influences of different GNNs. In particular, we use hierarchical graph pooling methods as global graph pooling methods in this experiment, including $\text{SOP}\textsc{ool}_{m\_attn}$.

- Is the improvement brought by our proposed method consistent with various GNN architectures? Section 5.4.4 shows the performance of the proposed $\text{SOP}\textsc{ool}_{bimap}$ and $\text{SOP}\textsc{ool}_{attn}$ with different GNNs.

- Is $\text{SOP}\textsc{ool}_{m\_attn}$ effective as hierarchical graph pooling methods? In Section 5.4.5, we compare $\text{SOP}\textsc{ool}_{m\_attn}$ with other hierarchical graph pooling methods in the same hierarchical GNN architecture.

5.4.1 Experimental Setup

Reproducibility. The code used in our experiments is available at `https://github.com/divelab/sopool`. Details of datasets and parameter settings are described below.

Datasets. We use ten graph classification datasets from [131], including five bioinformatics datasets (MUTAG, PTC, PROTEINS, NCI1, DD) and five social network datasets (COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI5K). Note that only bioinformatics datasets come with node labels. Below are the detailed descriptions of datasets:

- MUTAG is a bioinformatics dataset of 188 graphs representing nitro compounds. Each node is associated with one of 7 discrete node labels. The task is to classify each graph by determining whether the compound is mutagenic aromatic or heteroaromatic [157].
- PTC [158] is a bioinformatics dataset of 344 graphs representing chemical compounds. Each node comes with one of 19 discrete node labels. The task is to predict the rodent carcinogenicity for each graph.
- PROTEINS [159] is a bioinformatics dataset of 1,113 graph structures of proteins. Nodes in the graphs refer to secondary structure elements (SSEs) and have discrete node labels indicating whether they represent a helix, sheet or turn. And edges mean that two nodes are neighbors along the amino-acid sequence or in space. The task is to predict the protein function for each graph.
- NCI1 [160] is a bioinformatics dataset of 4,110 graphs representing chemical compounds. It contains data published by the National Cancer Institute (NCI). Each node is assigned with one of 37 discrete node labels. The graph classification label is decided by NCI anti-cancer screens for ability to suppress or inhibit the growth of a panel of human tumor cell lines.
- COLLAB is a scientific collaboration dataset of 5,000 graphs corresponding to ego-networks generated using the method in [161]. The dataset is derived from 3 public collaboration datasets [162]. Each ego-network contains different researchers from each field and is labeled by the corresponding field. The three fields are High Energy Physics, Condensed Matter Physics, and Astro Physics.

- IMDB-BINARY is a movie collaboration dataset of 1,000 graphs representing ego-networks for actors/actresses. The dataset is derived from collaboration graphs on Action and Romance genres. In each graph, nodes represent actors/actresses and edges simply mean they collaborate the same movie. The graphs are labeled by the corresponding genre and the task is to identify the genre for each graph.

- IMDB-MULTI is multi-class version of IMDB-BINARY. It contains 1,500 ego-networks and has three extra genres, namely, Comedy, Romance and Sci-Fi.

- REDDIT-BINARY is a dataset of 2,000 graphs where each graph represents an online discussion thread. Nodes in a graph correspond to users appearing in the corresponding discussion thread and an edge means that one user responded to another. Datasets are crawled from top submissions under four popular subreddits, namely, IAmA, AskReddit, TrollXChromosomes, atheism. Among them, AmA and AskReddit are question/answer-based subreddits while TrollXChromosomes and atheism are discussion-based subreddits, forming two classes to be classified.

- REDDIT-MULTI5K is a similar dataset as REDDIT-BINARY, which contains 5,000 graphs. The difference lies in that REDDIT-MULTI5K crawled data from five different subreddits, namely, worldnews, videos, AdviceAnimals, aww and mildlyinteresting. And the task is to identify the subreddit of each graph instead of determining the type of subreddits.

- DD [163] is a bioinformatics dataset of 1,178 graph structures of proteins. Nodes in the graphs represent amino acids. And edges connect nodes that are less than 6 $\mathring{A}$ngstroms apart. The task is a two-way classification task between enzymes and non-enzymes. DD is only used in Section 5.4.5. The average number of nodes in DD is 284.3.

More statistics of these datasets are provided in the "datasets" section of Table 5.1. The input node features are different for different datasets. For bioinformatics datasets, the nodes have categorical labels as input features. For social network datasets, we create node features. To be specific, we set all node feature vectors to be the same for REDDIT-BINARY and REDDIT-MULTI5K [3]. And for the other social network datasets, we use one-hot encoding of node degrees as features.

Configurations. In Sections 5.4.2, 5.4.3 and 5.4.4, the flat GNNs we use with our proposed graph pooling methods are graph isomorphism networks (GINs) [3]. The original GINs employ averaging or summation (SUM/AVG) as the graph pooling function; specifically, summation on bioinformatics datasets and averaging on social datasets. We replace averaging or summation with our proposed graph pooling methods and keep other parts the same. There are seven variants of GINs, two of which are equivalent to graph convolutional network (GCN) [15] and GraphSAGE [16], respectively. In Sections 5.4.2 and 5.4.3, we use GIN-0 with our methods. In Section 5.4.4, we examine our methods with all variants of GINs. Details of all variants can be found in Section 5.4.4.

The hierarchical GNNs used in Section 5.4.5 follow the hierarchical architecture in [6], allowing direct comparisons. To be specific, each block is composed of one GNN layer followed by a hierarchical graph pooling. After each hierarchical pooling, a classifier is used. The final prediction is the combination of all classifiers.

Training & Evaluation. Following [131, 164], model performance is evaluated using 10-fold cross-validation and reported as the average and standard deviation of validation accuracies across the 10 folds. For the flat GNNs, we follow the same training process in [3]. All GINs have 5 layers. Each multi-layer perceptron (MLP) has 2 layers with batch normalization [75]. For the hierarchical GNNs, we follow the the same training process in [6]. There are three blocks in total. Dropout [76] is applied in the classifiers. The Adam optimizer [84] is used with the learning rate initialized as 0.01 and decayed by 0.5 every 50 epochs. The number of total epochs is selected according to the best cross-validation accuracy. We tune the number of hidden units (16, 32, 64) and the batch size (32, 128) using grid search.

Baselines. We compare our methods with various graph classification models as baselines, including both kernel-based and GNN-based methods. The kernel-based methods are graphlet kernel (GK) [165], random walk kernel (RW) [166], Weisfeiler-Lehman subtree kernel (WL) [167],

Table 5.1: Comparison results between our proposed methods and baselines described in Section 5.4.1. We report the accuracies of these baselines provided in [1, 2, 3, 4, 5]. The best models are highlighted with boldface. If a kernel-based baseline performs the best than all GNN-based models, we highlight the best GNN-based model with boldface and the best kernel-based baseline with boldface and asterisk. ©2020 IEEE

		MUTAG	PTC	PROTEINS	NCI1	COLLAB	IMDB-B	IMDB-M	RDT-B	RDT-M5K
Datasets	# graphs	188	344	1113	4110	5000	1000	1500	2000	5000
	# classes	2	2	2	2	3	2	3	2	5
	# nodes (max)	28	109	620	111	492	136	89	3783	3783
	# nodes (avg.)	18.0	25.6	39.1	29.9	74.5	19.8	13.0	429.6	508.5
Kernel	GK [2009]	81.4±1.7	57.3±1.4	71.7±0.6	62.3±0.3	72.8±0.3	65.9±1.0	43.9±0.4	77.3±0.2	41.0±0.2
	RW [2010]	79.2±2.1	57.9±1.3	74.2±0.4	>1 day	-	-	-	-	-
	WL [2011]	90.4±5.7	59.9±4.3	75.0±3.1	**86.0±1.8***	78.9±1.9	73.8±3.9	50.9±3.8	81.0±3.1	52.5±2.1
	DGK [2015]	-	60.1±2.6	75.7±0.5	80.3±0.5	73.1±0.3	67.0±0.6	44.6±0.5	78.0±0.4	41.3±0.2
	AWE [2018]	87.9±9.8	-	-	-	73.9±1.9	74.5±5.9	51.5±3.6	87.9±2.5	54.7±2.9
GNN	DCNN [2016]	67.0	56.6	61.3	56.6	52.1	49.1	33.5	-	-
	PATCHSCAN [2016]	92.6±4.2	60.0±4.8	75.9±2.8	78.6±1.9	72.6±2.2	71.0±2.2	45.2±2.8	86.3±1.6	49.1±0.7
	ECC [2017]	-	-	72.7	76.8	67.8	-	-	-	-
	DGCNN [2018]	85.8±1.7	58.6±2.5	75.5±1.0	74.4±0.5	73.8±0.5	70.0±0.9	47.8±0.9	76.0±1.7	48.7±4.5
	DIFFPOOL [2018]	80.6	-	76.3	76.0	75.5	-	-	-	-
	GCAPS-CNN [2018]	-	66.0±5.9	76.4±4.2	82.7±2.4	77.7±2.5	71.7±3.4	48.5±4.1	87.6±2.5	50.1±1.7
	GIN-0 + SUM/AVG [2018]	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	80.2±1.9	75.1±5.1	52.3±2.8	**92.4±2.5**	57.5±1.5
	EigenGCN [2019]	79.5	-	76.6	77.0	-	-	-	-	-
Ours	GIN-0 + SOPOOL$_{attn}$	93.6±4.1	72.9±6.2	79.4±3.2	82.8±1.4	**81.1±1.8**	78.1±4.0	54.3±2.6	91.7±2.7	58.3±1.4
	GIN-0 + SOPOOL$_{bimap}$	**95.3±4.4**	**75.0±4.3**	**80.1±2.7**	83.6±1.4	79.9±1.9	78.4±4.7	**54.6±3.6**	89.6±3.3	**58.4±1.6**
	GIN-0 + SOPOOL$_{m\_attn}$	95.2±5.4	74.4±5.5	79.5±3.1	**84.5±1.3**	77.6±1.9	**78.5±2.8**	54.3±2.1	90.0±0.8	55.8±2.2

deep graphlet kernel (DGK) [131], and anonymous walk embeddings (AWE) [168]. Among them, DGK and AWE use deep learning methods as well. The GNN-based methods are diffusion-convolutional neural network (DCNN) [169], PATCHSCAN [164], ECC [170], deep graph CNN (DGCNN) [1], differentiable pooling (DIFFPOOL) [4], graph capsule CNN (GCAPS-CNN) [2], self-attention graph pooling (SAGPOOL) [6], GIN [3], and eigenvector-based pooling (EigenGCN) [5]. We report the performance of these baselines provided in [1, 2, 3, 4, 6, 5].

5.4.2 Comparison with Baselines

The comparison results between our methods and baselines are reported in Table 5.1. GIN-0 equipped with our proposed graph pooling methods, *i.e.* "GIN-0 + SOPOOL$_{attn}$", "GIN-0 + SOPOOL$_{bimap}$", and "GIN-0 + SOPOOL$_{m\_attn}$", outperform all the baselines significantly on seven out of nine datasets. On NCI1, WL has better performance than all GNN-based models. However, "GIN-0 + SOPOOL$_{m\_attn}$" is the second best model and has improved performance over other

Table 5.2: Comparison results between our proposed methods and other graph pooling methods by fixing the GNN before graph pooling to GIN-0, as described in Section 5.4.3. The best models are highlighted with boldface. ©2020 IEEE

Models	MUTAG	PTC	PROTEINS	NCI1	COLLAB	IMDB-B	IMDB-M	RDT-B	RDT-M5K
GIN-0 + SUM/AVG	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	80.2±1.9	75.1±5.1	52.3±2.8	92.4±2.5	57.5±1.5
GIN-0 + DIFFPOOL	94.8±4.8	66.1±7.7	78.8±3.1	76.6±1.3	75.3±2.2	74.4±4.0	50.1±3.2	-	-
GIN-0 + SORTPOOL	95.2±3.9	69.5±6.3	79.2±3.0	78.9±2.7	78.2±1.6	77.5±2.7	53.1±2.9	81.6±4.6	48.4±4.8
GIN-0 + TOPKPOOL	94.7±3.5	68.4±6.4	79.1±2.2	79.6±1.7	79.6±2.1	77.8±5.1	53.7±2.8	-	-
GIN-0 + SAGPOOL	93.9±3.3	69.0±6.6	78.4±3.1	79.0±2.8	78.9±1.7	77.8±2.9	53.1±2.8	-	-
GIN-0 + ATTNPOOL	93.2±5.8	71.2±8.0	77.5±3.3	80.6±2.1	**81.8±2.2**	77.1±4.4	53.8±2.5	**92.5±2.3**	57.9±1.7
GIN-0 + SOPOOL$_{attn}$	93.6±4.1	72.9±6.2	79.4±3.2	82.8±1.4	81.1±1.8	78.1±4.0	54.3±2.6	91.7±2.7	58.3±1.4
GIN-0 + COVPOOL	**95.3±3.7**	73.3±5.1	**80.1±2.2**	83.5±1.9	79.3±1.8	72.1±5.1	47.8±2.7	90.3±3.6	**58.4±1.7**
GIN-0 + SOPOOL$_{bimap}$	**95.3±4.4**	**75.0±4.3**	**80.1±2.7**	83.6±1.4	79.9±1.9	78.4±4.7	**54.6±3.6**	89.6±3.3	**58.4±1.6**
GIN-0 + SOPOOL$_{m\_attn}$	95.2±5.4	74.4±5.5	79.5±3.1	**84.5±1.3**	77.6±1.9	**78.5±2.8**	54.3±2.1	90.0±0.8	55.8±2.2

GNN-based models. On REDDIT-BINARY, our methods achieve comparable performance to the best one.

It is worth noting that the baseline “GIN-0 + SUM/AVG” is the previous state-of-the-art model [3]. Our methods differ from it only in the graph pooling functions. The significant improvement demonstrates the effectiveness of our proposed graph pooling methods. In the next section, we compare our methods with other graph pooling methods by fixing the GNN before graph pooling to GIN-0, in order to eliminate the influences of different GNNs.

5.4.3 Ablation Studies in Flat Graph Neural Networks

We perform ablation studies to show that our proposed methods are superior to other global graph pooling methods under a fair setting. Starting from the baseline “GIN-0 + SUM/AVG”, we replace SUM/AVG with different graph pooling methods and keep all other configurations unchanged. The graph pooling methods we include are DIFFPOOL [4], SORTPOOL from DGCNN [1], TOPKPOOL from Graph U-Net [142], SAGPOOL [6], and COVPOOL and ATTNPOOL described in Section 5.3.5. DIFFPOOL, TOPKPOOL, and SAGPOOL are used as hierarchical graph pooling methods in their works, but they achieve good performance as global pooling methods as well [4, 6]. EIGENPOOL from EigenGCN suffers from significant performance loss as a global pooling method [5] so that we do not include it in the ablation studies. COVPOOL and ATTNPOOL use the same settings as our proposed methods.

Table 5.2 provides the comparison results. Our proposed SOPOOL$_{bimap}$ and SOPOOL$_{attn}$ achieve better performance than DIFFPOOL, SORTPOOL, TOPKPOOL, and SAGPOOL on all datasets, demonstrating the effectiveness of our graph pooling methods with second-order statistics.

To support our discussion in Section 5.3.5, we analyze the performance of COVPOOL and ATTNPOOL. Note that the same bilinear mapping technique used in SOPOOL$_{bimap}$ is applied on COVPOOL, in order to avoid the excessive number of parameters. COVPOOL achieves comparable performance to SOPOOL$_{bimap}$ on most datasets. However, huge performance loss is observed on PTC, IMDB-BINARY, and IMDB-MULTI, indicating that subtracting the mean is harmful in graph pooling.

Compared to SOPOOL$_{attn}$, ATTNPOOL suffers from performance loss on all datasets except COLLAB and REDDIT-BINARY. The loss is especially significant on bioinformatics datasets (PTC, PROTEINS, NCI1). However, ATTNPOOL achieves the best performance on COLLAB and REDDIT-BINARY among all graph pooling methods, although the added SOFTMAX function results in less discriminative power. The reason might be capturing the distributional information is more important than the exact structure in these datasets. It is similar to GINs, where using averaging as graph pooling achieves better performance on social network datasets than summation [3].

5.4.4 Results with Different Graph Neural Networks

We've already demonstrated the superiority of our proposed SOPOOL$_{bimap}$ and SOPOOL$_{attn}$ over previous pooling methods. Next, we show that their effectiveness is robust to different GNNs. In this experiment, we change GIN-0 into other six variants of GINs. Note that these variants cover Graph Convolutional Networks (GCN) [15] and GraphSAGE [16], thus including a wide range of different kinds of GNNs.

We first give details of different variants of graph isomorphism networks (GINs) [3]. Basically, GINs iteratively update the representation of each node in a graph by aggregating representations of its neighbors, where the iteration is achieved by stacking several layers. Therefore, it suffice to

Table 5.3: Results of our proposed methods with different GNNs before graph pooling, as described in Section 5.4.4. The architectures of different GNNs come from variants of GINs in [3], whose details can be found in the supplementary material. The best models are highlighted with boldface. ©2020 IEEE

GNNs	Pools	MUTAG	PTC	PROTEINS	NCI1	COLLAB	IMDB-B	IMDB-M
	Sum/Avg	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	80.2±1.9	75.1±5.1	52.3±2.8
Sum-MLP (GIN-0)	SOPool$_{attn}$	93.6±4.1	72.9±6.2	79.4±3.2	82.8±1.4	**81.1±1.8**	78.1±4.0	54.3±2.6
	SOPool$_{bimap}$	**95.3±4.4**	**75.0±4.3**	**80.1±2.7**	**83.6±1.4**	79.9±1.9	**78.4±4.7**	**54.6±3.6**
	Sum/Avg	89.0±6.0	63.7±8.2	75.9±3.8	82.7±1.6	80.1±1.9	74.3±5.1	52.1±3.6
Sum-MLP (GIN-ϵ)	SOPool$_{attn}$	92.6±5.4	**73.6±5.5**	79.2±1.9	83.1±1.8	**80.6±1.6**	**78.1±4.3**	**55.4±3.7**
	SOPool$_{bimap}$	**93.7±5.3**	73.5±7.0	**79.3±1.8**	**83.6±1.4**	80.4±2.4	77.5±4.5	54.5±3.5
	Sum/Avg	90.0±8.8	63.1±5.7	76.2±2.6	82.0±1.5	80.6±1.9	74.1±5.0	52.2±2.4
Sum-1-Layer	SOPool$_{attn}$	94.2±4.4	**73.6±6.5**	79.0±2.9	81.2±1.5	**81.2±1.6**	**78.6±4.1**	**54.5±3.0**
	SOPool$_{bimap}$	**95.8±4.2**	71.8±6.1	**80.1±2.5**	**82.4±1.3**	80.5±2.0	78.2±3.6	54.1±3.4
	Sum/Avg	83.5±6.3	66.6±6.9	75.5±3.4	80.9±1.8	79.2±2.3	73.7±3.7	52.3±3.1
Mean-MLP	SOPool$_{attn}$	**92.6±4.5**	**74.9±6.6**	**79.4±2.8**	80.6±1.1	80.0±2.0	77.5±3.9	**55.2±3.3**
	SOPool$_{bimap}$	90.4±6.2	72.7±4.0	79.3±2.4	**81.1±1.6**	**80.4±1.7**	**77.9±4.7**	55.0±3.7
	Sum/Avg	85.6±5.8	64.2±4.3	76.0±3.2	**80.2±2.0**	79.0±1.8	74.0±3.4	51.9±3.8
Mean-1-Layer (GCN)	SOPool$_{attn}$	90.0±5.1	**76.7±5.6**	78.5±2.8	78.0±1.8	80.2±1.6	**78.9±4.2**	**54.8±3.1**
	SOPool$_{bimap}$	**90.9±5.7**	70.9±4.1	**78.7±3.1**	78.8±1.1	**80.4±2.1**	77.7±4.5	54.5±4.0
	Sum/Avg	84.0±6.1	64.6±10.2	76.0±3.2	77.8±1.3	-	73.2±5.8	51.1±3.6
Max-MLP	SOPool$_{attn}$	**90.0±7.3**	72.4±4.7	78.3±3.1	**78.6±1.9**	-	78.1±4.1	54.1±3.4
	SOPool$_{bimap}$	88.8±7.0	**73.3±5.5**	**78.4±3.0**	78.0±1.9	-	**78.2±4.7**	**54.6±3.5**
	Sum/Avg	85.1±7.6	63.9±7.7	75.9±3.2	**77.7±1.5**	-	72.3±5.3	50.9±2.2
Max-1-Layer (GraphSAGE)	SOPool$_{attn}$	**90.0±6.8**	72.1±5.9	**79.0±2.9**	77.4±1.8	-	77.4±5.1	54.1±3.1
	SOPool$_{bimap}$	89.9±5.8	**73.6±5.1**	78.9±2.8	77.0±2.0	-	**78.6±4.7**	**54.2±3.9**

describe the k-th layer of GINs based on one node.

Recall that we represent a graph $G = (A, X)$ by its adjacency matrix $A \in \{0, 1\}^{n \times n}$ and node feature matrix $X \in \mathbb{R}^{n \times d}$, where n is the number of nodes in G and d is the dimension of node features. The adjacency matrix tells the neighboring information of each node. We introduce GINs by defining node representation matrices $H^{(k-1)} \in \mathbb{R}^{n \times f^{(k-1)}}$ and $H^{(k)} \in \mathbb{R}^{n \times f^{(k)}}$ as inputs and outputs to the k-th layer, respectively. We have $H^{(0)} = X$. Note that the first dimension n does not change during the computation, as GINs learn representations for each node.

Specifically, consider a node ν has corresponding representations $h_\nu^{(k-1)} \in \mathbb{R}^{f^{(k-1)}}$ and $h_\nu^{(k)} \in \mathbb{R}^{f^{(k)}}$, which are rows of $H^{(k-1)}$ and $H^{(k)}$, respectively. The set of neighboring nodes of ν is given by $\mathcal{N}(\nu)$. We describe the k-layer of the following variants:

- **SUM-MLP (GIN-0):**

$$h_\nu^{(k)} = \text{MLP}^{(k)}(h_\nu^{(k-1)} + \sum_{\mu \in \mathcal{N}(\nu)} h_\mu^{(k-1)})$$

- **SUM-MLP (GIN-ϵ):**

$$h_\nu^{(k)} = \text{MLP}^{(k)}((1 + \epsilon^{(k)})h_\nu^{(k-1)} + \sum_{\mu \in \mathcal{N}(\nu)} h_\mu^{(k-1)})$$

- **SUM-1-LAYER:**

$$h_\nu^{(k)} = \text{ReLU}(W^{(k)}(h_\nu^{(k-1)} + \sum_{\mu \in \mathcal{N}(\nu)} h_\mu^{(k-1)}))$$

- **MEAN-MLP:**

$$h_\nu^{(k)} = \text{MLP}^{(k)}(\text{MEAN}\{h_\mu^{(k-1)}, \forall \mu \in \nu \cup \mathcal{N}(\nu)\})$$

- **MEAN-1-LAYER (GCN):**

$$h_\nu^{(k)} = \text{ReLU}(W^{(k)}(\text{MEAN}\{h_\mu^{(k-1)}, \forall \mu \in \nu \cup \mathcal{N}(\nu)\})$$

- **MAX-MLP:**

$$h_\nu^{(k)} = \text{MLP}^{(k)}(\text{MAX}\{h_\mu^{(k-1)}, \forall \mu \in \nu \cup \mathcal{N}(\nu)\})$$

- **MAX-1-LAYER (GraphSAGE):**

$$h_\nu^{(k)} = \text{ReLU}(W^{(k)}(\text{MAX}\{h_\mu^{(k-1)}, \forall \mu \in \nu \cup \mathcal{N}(\nu)\})$$

Here, the multi-layer perceptron (MLP) has two layers with ReLU activation functions. Note that MEAN-1-LAYER and MAX-1-LAYER correspond to GCN [15] and GraphSAGE [16], respectively, up to minor architecture modifications.

The results of these different GNNs with our graph pooling methods are reported in Table 5.3. Our proposed SOPOOL$_{bimap}$ and SOPOOL$_{attn}$ achieve satisfying performance consistently. In particular, on social network datasets, the performance does not decline when the GNNs before graph pooling become less powerful, showing the highly discriminative ability of second-order pooling.

5.4.5 Ablation Studies in Hierarchical Graph Neural Networks

SOPOOL$_{m\_attn}$ has shown its effectiveness as global graph pooling through the experiments in Sections 5.4.2 and 5.4.3. In this section, we evaluate it as hierarchical graph pooling in hierarchical GNNs. The hierarchical GNN architecture follows the one in [6], which contains three blocks of a GNN layer followed by graph pooling, as introduced in Section 5.4.1. The experiments are performed on DD and PROTEINS datasets, where hierarchical GNNs tend to achieve good performance [4, 6].

First, we compare SOPOOL$_{m\_attn}$ with different hierarchical graph pooling methods under the same hierarchical GNN architecture. Specifically, we include DIFFPOOL, TOPKPOOL, and SAGPOOL, which have been used as hierarchical graph pooling methods in their works. The comparison results are provided in Table 5.4. Our proposed SOPOOL$_{m\_attn}$ outperforms all the baselines on both datasets, indicating the effectiveness of SOPOOL$_{m\_attn}$ as a hierarchical graph

Table 5.4: Comparison results between different hierarchical graph pooling methods. The hierarchical GNN architecture follows the one in [6]. We report the accuracies of the baselines provided in [6]. The best models are highlighted with boldface. ©2020 IEEE

Models	DD	PROTEINS
DiffPool	67.0±2.4	68.2±2.0
TopKPool	75.0±0.9	71.1±0.9
SAGPool	76.5±1.0	71.9±1.0
SOPool$_{m\_attn}$	**76.8**±**1.9**	**77.1**±**3.8**

Table 5.5: Comparison results of SOPool$_{m\_attn}$ with different hierarchical GNNs. The hierarchical GNN architecture follows the one in [6], where we change the number of blocks from one to three. The best models are highlighted with boldface. ©2020 IEEE

Models	DD	PROTEINS
1 block	73.3±2.4	77.4±4.3
2 blocks	**77.2**±**2.7**	**78.1**±**4.3**
3 blocks	76.8±1.9	77.1±3.8

pooling method.

In addition, we conduct experiments to evaluate SOPool$_{m\_attn}$ in different hierarchical GNNs by varying the number of blocks. The results are shown in Table 5.5. On both datasets, our SOPool$_{m\_attn}$ achieves the best performance when the number of blocks is two. The results indicate current datasets on graph classification are not large enough yet. And without techniques like jumping knowledge networks (JK-Net) [138], hierarchical GNNs tend to suffer from overfitting, leading to worse performance than flat GNNs.

6. DEEP ATTENTION NETWORKS FOR GRAPHS: NON-LOCAL GRAPH NEURAL NETWORKS

In this chapter, we propose the non-local GNNs for disassortative graphs, addressing the problem that modern GNNs lack the ability of performing non-local aggregation. Non-local aggregation is essential for tasks on disassortative graphs. In particular, we propose a simple yet effective non-local aggregation framework with an efficient attention-guided sorting for GNNs. Based on it, we develop various non-local GNNs.

6.1 Introduction

Graph neural networks (GNNs) process graphs and map each node to an embedding vector [132, 133]. These node embeddings can be directly used for node-level applications, such as node classification [15] and link prediction [139]. In addition, they can be used to learn the graph representation vector with graph pooling [4, 1, 6, 171], in order to fit graph-level tasks [131]. Many variants of GNNs have been proposed, such as ChebNets [141], GCNs [15], GraphSAGE [16], GATs [17], LGCN [18] and GINs [3]. Their advantages have been shown on various graph datasets and tasks [172]. However, these GNNs share a multilayer local aggregation framework, which is similar to convolutional neural networks (CNNs) [8] on grid-like data such as images and texts.

In recent years, the importance of non-local aggregation has been demonstrated in many applications in the field of computer vision [70, 28] and natural language processing [13]. In particular, the attention mechanism has been widely explored to achieve non-local aggregation and capture long-range dependencies from distant locations. Basically, the attention mechanism measures the similarity between every pair of locations and enables information to be communicated among distant but similar locations. In terms of graphs, non-local aggregation is also crucial for disassortative graphs, while previous studies of GNNs focus on assortative graph datasets (Section 6.2.2). In addition, we find that local aggregation is even harmful for some disassortative graphs (Section 6.4.2). The recently proposed Geom-GCN [173] explores to capture long-range dependencies in disassortative graphs. It contains an attention-like step that computes the Euclidean distance between every pair of nodes. However, this step is computationally prohibitive for large-scale graphs, as the computational complexity is quadratic in the number of nodes. In addition, Geom-GCN employs pre-trained node embeddings [174, 175, 176] that are not task-specific, limiting the effectiveness and flexibility.

In this chapter, we propose a simple yet effective non-local aggregation framework for GNNs. At the heart of the framework lies an efficient attention-guided sorting, which enables non-local aggregation through classic local aggregation operators in general deep learning. The proposed framework can be flexibly used to augment common GNNs with low computational costs. Based on the framework, we build various efficient non-local GNNs. In addition, we perform detailed analysis on existing disassortative graph datasets, and apply different non-local GNNs accordingly. Experimental results show that our non-local GNNs significantly outperform previous state-of-the-art methods on node classification tasks on six benchmark datasets of disassortative graphs.

6.2 Related Work

6.2.1 Graph Neural Networks

We focus on learning the embedding vector for each node through graph neural networks (GNNs). Most existing GNNs are inspired by the success of convolutional neural networks (CNNs) [8] and follow a local aggregation framework. In general, each layer of GNNs scans every node in the graph and aggregates local information from directly connected nodes, *i.e.*, the 1-hop neighbors.

Specifically, a common layer of GNNs performs a two-step processing similar to the depthwise separable convolution [56]: spatial aggregation and feature transformation. The first step updates each node embedding using embedding vectors of spatially neighboring nodes. For example, GCNs [15] and GATs [17] compute a weighted sum of node embeddings within the 1-hop neighborhood, where weights come from the degree of nodes and the interaction between nodes, respectively. GraphSAGE [16] applies the max pooling, while GINs [3] simply sums the node embeddings. The feature transformation step is similar to the 1×1 convolution, where each node embedding vector is mapped into a new feature space through a shared linear transformation [15, 16, 17] or multilayer perceptron (MLP) [3]. Different from these studies, LGCN [18] explores to directly apply the regular convolution through top-k ranking.

Nevertheless, each layer of these GNNs only aggregates local information within the 1-hop neighborhood. While stacking multiple layers can theoretically enable communication between nodes across the multi-hop neighborhood, the aggregation is essentially local. In addition, deep GNNs usually suffer from the over-smoothing problem [138, 177, 144].

6.2.2 Assortative and Disassortative Graphs

There are many kinds of graphs in the literature, such as citation networks [15], community networks [144], co-occurrence networks [178], and webpage linking networks [179]. We focus on graph datasets corresponding to the node classification tasks. In particular, we categorize graph datasets into assortative and disassortative ones [180, 176] according to the node homophily in terms of labels, *i.e.*, how likely nodes with the same label are near each other in the graph.

Assortative graphs refer to those with a high node homophily. Common assortative graph datasets are citation networks and community networks. On the other hand, graphs in disassortative graph datasets contain more nodes that have the same label but are distant from each other. Example disassortative graph datasets are co-occurrence networks and webpage linking networks.

As introduced above, most existing GNNs perform local aggregation only and achieve good performance on assortative graphs [15, 16, 17, 18]. However, they may fail on disassortative graphs, where informative nodes in the same class tend to be out of the local multi-hop neighborhood and non-local aggregation is needed. Thus, in this chapter, we explore the non-local GNNs.

6.2.3 Attention Mechanism

The attention mechanism [13] has been widely used in GNNs [17, 181, 182] as well as other deep learning models [26, 70, 28]. A typical attention mechanism takes three groups of vectors as inputs, namely the query vector q, key vectors $(k_1, k_2, \ldots, k_n)$, value vectors $(v_1, v_2, \ldots, v_n)$. Note that key and value vectors have a one-to-one correspondence and can be the same sometimes. The attention mechanism computes the output vector o as

$$a_i = \textsc{Attend}(q, k_i) \in \mathbb{R},\ i = 1, 2, \ldots, n; \quad o = \sum_i a_i v_i, \tag{6.1}$$

where the $\textsc{Attend}(\cdot)$ function could be any function that outputs a scalar attention score a_i from the interaction between q and k_i, such as dot product [181] or even a neural network [17]. The definition of the three groups of input vectors depends on the models and applications.

Notably, existing GNNs usually use the attention mechanism for local aggregation [17, 181]. Specifically, when aggregating information for node v, the query vector is the embedding vector of v while the key and value vectors come from node embeddings of v's directly connected nodes. And the process is iterated for each $v \in V$. It is worth noting that the attention mechanism can be easily extended for non-local aggregation [70, 28], by letting the key and value vectors correspond to all the nodes in the graph when aggregating information for each node. However, it is computationally prohibitive given large-scale graphs, as iterating it for each node in a graph of n nodes requires $O(n^2)$ time. In this chapter, we propose a novel non-local aggregation method that only requires $O(n \log n)$ time.

6.3 Non-local Graph Neural Networks

6.3.1 Non-local Aggregation with Attention-guided Sorting

We consider a graph $\mathcal{G} = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge $e \in E$ connects two nodes so that $E \subset V \times V$. Each node $v \in V$ has a corresponding node feature vector $x_v \in \mathbb{R}^d$. The k-hop neighborhood of v refers to the set of nodes $\mathcal{N}_k(v)$ that can reach v within k edges. For example, the set of v's directly connected nodes is its 1-hop neighborhood $\mathcal{N}_1(v)$.

Our proposed non-local aggregation framework is composed of three steps, namely local embedding, attention-guided sorting, and non-local aggregation. In the following, we describe them one by one.

Local Embedding: Our proposed framework is built upon a local embedding step that extracts local node embeddings from the node feature vectors. The local embedding step can be as simple as

$$z_v = \text{MLP}(x_v) \in \mathbb{R}^f,\ \forall v \in V, \tag{6.2}$$

where the MLP$(\cdot)$ function refers to a multilayer perceptron (MLP), and f is the dimension of the local node embedding z_v. Note that the MLP$(\cdot)$ function is shared across all the nodes in the graph. Applying MLP only takes the node itself into consideration without aggregating information from the neighborhood. The property is very important on some disassortative graphs, as shown in Section 6.4.2.

On the other hand, graph neural networks (GNNs) can be used as the local embedding step as well, so that our proposed framework can be easily employed to augment existing GNNs. As introduced in Section 6.2.1, modern GNNs perform multilayer local aggregation. Typically, for each node, one layer of a GNN aggregates information from its 1-hop neighborhood. Stacking L such local aggregation layers allows each node to access information that is L hops away. To be specific, the l-th layer of a L-layer GNN $(l = 1, 2, \dots, L)$ can be described as

$$z_v^{(l)} = \textsc{Transform}^{(l)}\left(\textsc{Aggregate}^{(l)}\left(\{z_u^{(l-1)} : u \in \mathcal{N}_1(v) \cup v\}\right)\right) \in \mathbb{R}^f,\ \forall v \in V, \tag{6.3}$$

where $z_v^{(0)} = x_v$, and $z_v = z_v^{(L)}$ represents the local node embedding. The $\textsc{Aggregate}^{(l)}(\cdot)$ and $\textsc{Transform}^{(l)}(\cdot)$ functions represent the spatial aggregation and feature transformation step introduced in Section 6.2.1, respectively. With the above framework, GNNs can capture the node feature information from nodes within a local neighborhood as well as the structural information.

When either MLP or GNNs is used as the local embedding step, the local node embedding z_v only contains local information of a node v. However, z_v can be used to guide non-local aggregation, as distant but informative nodes are likely to have similar node features and local structures. Based on this intuition, we propose the attention-guided sorting to enable the non-local aggregation.

Attention-guided Sorting: The basic idea of the attention-guided sorting is to learn an ordering of nodes, where distant but informative nodes are put near each other. Specifically, given the local node embedding z_v obtained through the local embedding step, we compute one set of attention scores by

$$a_v = \textsc{Attend}(c, z_v) \in \mathbb{R},\ \forall v \in V, \tag{6.4}$$

where c is a calibration vector that is randomly initialized and jointly learned during training [26]. In this attention operator, c serves as the query vector and z_v are the key vectors. In addition, we also treat z_v as the value vectors. However, unlike the attention mechanism introduced in Section 6.2.3, we use the attention scores to sort the value vectors instead of computing a weighted sum to aggregating them. Note that originally there is no ordering among nodes in a graph. To be specific, as a_v and z_v have one-to-one correspondence through Equation (6.4), sorting the attention scores in non-decreasing order into $(a_1, a_2, \ldots, a_n)$ provides an ordering among nodes, where $n = |V|$ is the number of nodes in the graph. The resulting sequence of local node embeddings can be denoted as $(z_1, z_2, \ldots, z_n)$.

The attention process in Equation (6.4) can be also understood as a projection of local node embeddings onto a 1-dimensional space. The projection depends on the concrete $\textsc{Attend}(\cdot)$ function and the calibration vector c. As indicated by its name, the calibration vector c is used to calibrate the 1-dimensional space, in order to push distant but informative nodes close to each other in this space. This goal is fulfilled through the following non-local aggregation step and the training of the calibration vector c, as demonstrated below.

Non-local Aggregation: We point out that, with the attention-guided sorting, the non-local aggregation can be achieved by convolution, the most common local aggregation operator in deep learning. Specifically, given the sorted sequence of local node embeddings $(z_1, z_2, \ldots, z_n)$, we compute

$$(\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_n) = \textsc{Conv}(z_1, z_2, \ldots, z_n), \tag{6.5}$$

where the $\textsc{Conv}(\cdot)$ function represents a 1D convolution with appropriate padding. Note that the

$\textsc{Conv}(\cdot)$ function can be replaced by a 1D convolutional neural network as long as the number of input and output vectors remains the same.

To see how the $\textsc{Conv}(\cdot)$ function performs non-local aggregation with the attention-guided sorting, we take an example where the $\textsc{Conv}(\cdot)$ function is a 1D convolution of kernel size $2s+1$. In this case, $\hat{z}_i$ is computed from $(z_{i+s}, \ldots, z_{i-s})$, corresponding to the receptive field of the $\textsc{Conv}(\cdot)$ function. As a result, if the attention-guided sorting leads to $(z_{i+s}, \ldots, z_{i-s})$ containing nodes that are distant but informative to z_i, the output $\hat{z}_i$ aggregates non-local information. Another view is that we can consider the attention-guided sorting as re-connects nodes in the graph, where $(z_{i+s}, \ldots, z_{i-s})$ can be treated as the 1-hop neighborhood of z_i. After the $\textsc{Conv}(\cdot)$ function, $\hat{z}_i$ and z_i are concatenated as the input to a classifier to predict the label of the corresponding node, where both non-local and local dependencies can be captured. In order to enable the end-to-end training of the calibration vector c, we modify Equation (6.5) into

$$(\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_n) = \textsc{Conv}(a_1 z_1, a_2 z_2, \ldots, a_n z_n), \tag{6.6}$$

where we multiply the attention score with the corresponding local node embedding. As a result, the calibration vector c receives gradients through the attention scores during training.

The remaining question is how to make sure that the attention-guided sorting pushes distant but informative nodes together. The short answer is that it is not necessary to guarantee this, as the requirement of non-local aggregation depends on the concrete graphs. In fact, our proposed framework grants GNNs the ability of non-local aggregation but lets the end-to-end training process determine whether to use non-local information. The back-propagation from the supervised loss will tune the calibration vector c and encourage $\hat{z}_i$ to capture useful information that is not encoded by z_i. In the case of disassortative graphs, $\hat{z}_i$ usually needs to aggregate information from distant but informative nodes. Hence, the calibration vector c tends to arrange the attention-guided sorting to put distant but informative nodes together. On the other hand, nodes within the local neighborhood are usually much more informative than distant nodes in assortative graphs. In this situation, $\hat{z}_i$ may simply perform local aggregation that is similar to GNNs.

In Section 6.4, we demonstrate the effectiveness of our proposed non-local aggregation framework on six disassortative graph datasets. In particular, we achieve the state-of-the-art performance on all the datasets with significant improvements over previous methods.

6.3.2 Time Complexity Analysis

We perform theoretical analysis of the time complexity of our proposed framework. As discussed in Section 6.2.3, using the attention mechanism [13, 70, 28] to achieve non-local aggregation requires $O(n^2)$ time for a graph of n nodes. Essentially, the $O(n^2)$ time complexity is due to the fact that the $\text{ATTEND}(\cdot)$ function needs to be computed between every pair of nodes. In particular, the recently proposed Geom-GCN [173] contains a similar non-local aggregation step. For each $v \in V$, Geom-GCN finds the set of nodes from which the Euclidean distance to v is less than a pre-defined number, where the Euclidean distance between every pair of nodes needs to be computed. As the computation of the the Euclidean distance between two nodes can be understood as the $\text{ATTEND}(\cdot)$ function, Geom-GCN has at least $O(n^2)$ time complexity.

In contrast, our proposed non-local aggregation framework requires only $O(n \log n)$ time. To see this, note that the $\text{ATTEND}(\cdot)$ function in Equation (6.4) only needs to be computed once, instead of iterating it for each node. As a result, computing the attention scores only takes $O(n)$ time. Therefore, the time complexity of sorting, *i.e.* $O(n \log n)$, dominates the total time complexity of our proposed framework. In Section 6.4.5, we compare the real running time on different datasets among common GNNs, Geom-GCN, and our non-local GNNs as introduced in the next section.

6.3.3 Efficient Non-local Graph Neural Networks

We apply our proposed non-local aggregation framework to build efficient non-local GNNs. Recall that our proposed framework starts with the local embedding step, followed by the attention-guided sorting and the non-local aggregation step.

In particular, the local embedding step can be implemented by either MLP or common GNNs, such as GCNs [15] or GATs [17]. MLP extracts the local node embedding only from the node feature vector and excludes the information from nodes within the local neighborhood. This property can be helpful on some disassortative graphs, where nodes within the local neighborhood provide more noises than useful information. On other disassortative graphs, informative nodes locate in both local neighborhood and distant locations. In this case, GNNs are more suitable as the local embedding step. Depending on the disassortative graphs in hand, we build different non-local GNNs with either MLP or GNNs as the local embedding step. In Section 6.4.2, we show that these two categories of disassortative graphs can be distinguished through simple experiments, where we apply different non-local GNNs accordingly. Specifically, the number of layers is set to 2 for both MLP and GNNs.

In terms of the attention-guided sorting, we only need to specify the $\text{ATTEND}(\cdot)$ function in Equation (6.4). In order to make it as efficient as possible, we choose the simplest $\text{ATTEND}(\cdot)$ function as

$$a_v = \text{ATTEND}(c, z_v) = c^T z_v \in \mathbb{R},\ \forall v \in V, \tag{6.7}$$

where c is part of the training parameters, as described in Section 6.3.1.

With the the attention-guided sorting, we can implement the non-local aggregation step through convolution, as explained in Section 6.3.1 and shown in Equation (6.6). Specifically, we set the $\text{CONV}(\cdot)$ function to be a 2-layer convolutional neural network composed of two 1D convolutions. The kernel size is set to 3 or 5 depending on the datasets. The activation function is ReLU [9].

Finally, we use a linear classifier that takes the concatenation of $\hat{z}_i$ and z_i as inputs and makes prediction for the corresponding node. Depending on the local embedding step, we build three efficient non-local GNNs, namely non-local MLP (NLMLP), non-local GCN (NLGCN), and non-local GAT (NLGAT). The models can be end-to-end trained with the classification loss.

Table 6.1: Statistics of the nine datasets used in our experiments. The definition of $H(\mathcal{G})$ is provided in Section 6.4.1.1. $H(\mathcal{G})$ can be used to distinguish assortative and disassortative graph datasets.

	Assortative			**Disassortative**					
Datasets	*Cora*	*Citeseer*	*Pubmed*	*Chameleon*	*Squirrel*	*Actor*	*Cornell*	*Texas*	*Wisconsin*
$H(\mathcal{G})$	0.83	0.71	0.79	0.25	0.22	0.24	0.11	0.06	0.16
#Nodes	2708	3327	19717	2277	5201	7600	183	183	251
#Edges	5429	4732	44338	36101	217073	33544	295	309	499
#Features	1433	3703	500	2325	2089	931	1703	1703	1703
#Classes	7	6	3	5	5	5	5	5	5

6.4 Experimental Studies

6.4.1 Experimental Setup

6.4.1.1 Datasets

We perform experiments on six disassortative graph datasets [179, 178, 173] (*Chameleon*, *Squirrel*, *Actor*, *Cornell*, *Texas*, *Wisconsin*) and three assortative graph datasets [15] (*Cora*, *Citeseer*, *Pubmed*). These datasets are commonly used to evaluate GNNs on node classification tasks [15, 17, 18, 173]. In order to distinguish assortative and disassortative graph datasets, Pei et al. [173] propose a metric to measure the homophily of a graph $\mathcal{G}$, defined as

$$H(\mathcal{G}) = \frac{1}{|V|} \sum_{v \in V} \frac{\text{Number of } v\text{'s directly connected nodes who have the same label as } v}{\text{Number of } v\text{'s directly connected nodes}}. \quad (6.8)$$

Intuitively, a large $H(\mathcal{G})$ indicates an assortative graph, and vice versa. The $H(\mathcal{G})$ and other statistics are summarized in Table 6.1.

In our experiments, we focus on comparing the model performance on disassortative graph datasets, in order to demonstrate the effectiveness of our non-local aggregation framework. The performances on assortative graph datasets are provided for reference, indicating that the proposed framework will not hurt the performance when non-local aggregation is not strongly desired.

6.4.1.2 Baselines

We compare our proposed non-local MLP (NLMLP), non-local GCN (NLGCN), and non-local GAT (NLGAT) with various baselines:

- MLP is the simplest deep learning model. It makes prediction solely based on the node feature vectors, without aggregating any local or non-local information.
- GCN [15] and GAT [17] are the most common GNNs. As introduced in Section 6.2.1, they only perform local aggregation.
- Geom-GCN [173] is a recently proposed GNN that can capture long-range dependencies. It is the current state-of-the-art model on several disassortative graph datasets. Geom-GCN requires the use of different node embedding methods, such as Isomap [174], Poincare [175], and struc2vec [176]. We simply report the best results from [173] for Geom-GCN and the following two variants without specifying the node embedding method.
- Geom-GCN-g [173] is a variant of Geom-GCN that performs local aggregation only. It is similar to common GNNs.
- Geom-GCN-s [173] is a variant of Geom-GCN that does not force local aggregation. The designed functionality is similar to our NLMLP.

We implement MLP, GCN, GAT, and our methods using Pytorch [183] and Pytorch Geometric [184]. As has been discussed*, in fair settings, the results of GCN and GAT differ from those in [173].

On each dataset, we follow [173] and randomly split nodes of each class into 60%, 20%, and 20% for training, validation, and testing. The experiments are repeatedly run 10 times with different random splits and the average test accuracy over these 10 runs are reported. Testing is performed when validation accuracy achieves maximum on each run. Apart from the details specified in Section 6.3.3, we tune the following hyperparameters individually for our proposed

*`https://openreview.net/forum?id=S1e2agrFvS&noteId=8tGKV1oSzCr`

Table 6.2: Comparisons between MLP and common GNNs (GCN, GAT). These analytical experiments are used to determine the two categories of disassortative graph datasets, as introduced in Section 6.4.2.

	Assortative			**Disassortative**					
Datasets	*Cora*	*Citeseer*	*Pubmed*	*Chameleon*	*Squirrel*	*Actor*	*Cornell*	*Texas*	*Wisconsin*
MLP	76.5	73.6	87.5	48.5	31.5	**35.1**	**81.6**	**81.3**	**84.9**
GCN	88.2	75.7	**88.4**	**67.6**	**54.9**	30.3	54.2	61.1	59.6
GAT	**88.4**	**76.1**	87.0	64.9	51.3	29.4	56.3	57.9	57.8

models: (1) the number of hidden unit $\in \{16, 48, 96\}$, (2) dropout rate $\in \{0, 0.5, 0.8\}$, (3) weight decay $\in \{0, 5e\text{-}4, 5e\text{-}5, 5e\text{-}6\}$, and (4) learning rate $\in \{0.01, 0.05\}$.

6.4.2 Analysis of Disassortative Graph Datasets

As discussed in Section 6.3.3, the disassortative graph datasets can be divided into two categories. Nodes within the local neighborhood provide more noises than useful information in disassortative graphs belonging to the first category. Therefore, local aggregation should be avoided in models on such disassortative graphs. As for the second category, informative nodes locate in both local neighborhood and distant locations. Intuitively, a graph with lower $H(\mathcal{G})$ is more likely to be in the first category. However, it is not an accurate way to determine the two categories.

Knowing the exact category of a disassortative graph is crucial, as we need to apply non-local GNNs accordingly. As analyzed above, the key difference lies in whether the local aggregation is useful. Based on this insight, we can distinguish two categories of disassortative graph datasets by comparing the performance between MLP and common GNNs (GCN, GAT) on each of the six disassortative graph datasets.

The results are summarized in Table 6.2. We can see that *Actor*, *Cornell*, *Texas*, and *Wisconsin* fall into the first category, while *Chameleon* and *Squirrel* belong to the second category. We add the performance on assortative graph datasets for reference, where the local aggregation is effective so that GNNs tend to outperform MLP.

6.4.3 Comparisons with Baselines

According to the insights from Section 6.4.2, we apply different non-local GNNs according to the category of disassortative graph datasets, and make comparisons with corresponding baselines. Meanwhile, we still provide the results of all the models on all datasets in Table 6.3 for reference.

Table 6.3: Comparisons between our NLMLP, NLGCN, NLGAT and baselines on all the nine datasets.

	Assortative			**Disassortative**					
Datasets	*Cora*	*Citeseer*	*Pubmed*	*Chameleon*	*Squirrel*	*Actor*	*Cornell*	*Texas*	*Wisconsin*
MLP	76.5	73.6	87.5	48.5	31.5	35.1	81.6	81.3	84.9
GCN	88.2	75.7	88.4	67.6	54.9	30.3	54.2	61.1	59.6
GAT	88.4	76.1	87.0	64.9	51.3	29.4	56.3	57.9	57.8
Geom-GCN	85.3	78.0	90.1	60.9	38.1	31.6	60.8	67.6	64.1
Geom-GCN-g	87.0	**80.6**	**90.7**	68.0	46.0	32.0	67.0	73.1	69.4
Geom-GCN-s	73.3	72.2	87.0	61.6	38.0	34.6	75.4	73.5	80.4
NLMLP	76.9	73.4	88.2	50.7	33.7	**37.9**	**84.9**	**85.4**	**87.3**
NLGCN	88.1	75.2	89.0	**70.1**	**59.0**	31.6	57.6	65.5	60.2
NLGAT	**88.5**	76.2	88.2	65.7	56.8	29.5	54.7	62.6	56.9

Table 6.4: Comparisons between our NLMLP and strong baselines on the four disassortative graph datasets belonging to the first category as defined in Section 6.4.2.

Datasets	*Actor*	*Cornell*	*Texas*	*Wisconsin*
MLP	35.1	81.6	81.3	84.9
Geom-GCN	31.6	60.8	67.6	64.1
Geom-GCN-s	34.6	75.4	73.5	80.4
NLMLP	**37.9**	**84.9**	**85.4**	**87.3**

Specifically, we employ NLMLP on *Actor*, *Cornell*, *Texas*, and *Wisconsin*. The corresponding baselines are MLP, Geom-GCN, and Geom-GCN-s, as Table 6.2 has shown that GCN and GAT perform much worse than MLP on these datasets. And Geom-GCN-g is similar to GCN and has worse performance than Geom-GCN-s, which is shown in Table 6.3. The comparison results are reported in Table 6.4. While Geom-GCN-s are the previous state-of-the-art GNNs on these datasets [173], we find that MLP consistently outperforms Geom-GCN-s by large margins. In particular, although Geom-GCN-s does not explicitly perform local aggregation, it is still outperformed by MLP. A possible explanation is that Geom-GCN-s uses pre-trained node embeddings, which aggregates information from the local neighborhood implicitly. In contrast, our NLMLP is built upon MLP with the proposed non-local aggregation framework, which excludes the local noises and collects useful information from non-local informative nodes. The NLMLP sets the new state-of-the-art performance on these disassortative graph datasets.

Table 6.5: Comparisons between our NLGCN, NLGAT and strong baselines on the two disassortative graph datasets belonging to the second category as defined in Section 6.4.2.

Datasets	*Chameleon*	*Squirrel*
GCN	67.6	54.9
GAT	64.9	51.3
Geom-GCN	60.9	38.1
Geom-GCN-g	68.0	46.0
NLGCN	**70.1**	**59.0**
NLGAT	65.7	56.8

On *Chameleon* and *Squirrel* that belong to the second category of disassortative graph datasets, we apply NLGCN and NLGAT accordingly. The baselines are GCN, GAT, Geom-GCN, and Geom-GCN-g. In these datasets, these baselines that explicitly perform local aggregation show advantages over MLP and Geom-GCN-s, as shown in Table 6.3. Table 6.5 summarizes the comparison results. Our proposed NLGCN achieves the best performance on both datasets. In addition, it is worth noting that our NLGCN and NLGAT are built upon GCN and GAT, respectively. They show improvements over their counterparts, which indicates that the advantages of our proposed non-local aggregation framework are general for common GNNs.

6.4.4 Analysis of the Attention-guided Sorting

We analyze the results of the attention-guided sorting in our proposed framework, in order to show that our non-local GNNs indeed perform non-local aggregation.

Suppose the attention-guided sorting leads to the sorted sequence $(z_1, z_2, \ldots, z_n)$, which goes through a convolution or CNN into $(\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_n)$. As discussed in Section 6.3.1, we can consider the sequence $(z_1, z_2, \ldots, z_n)$ as a re-connected graph $\hat{\mathcal{G}}$, where we treat nodes within the receptive

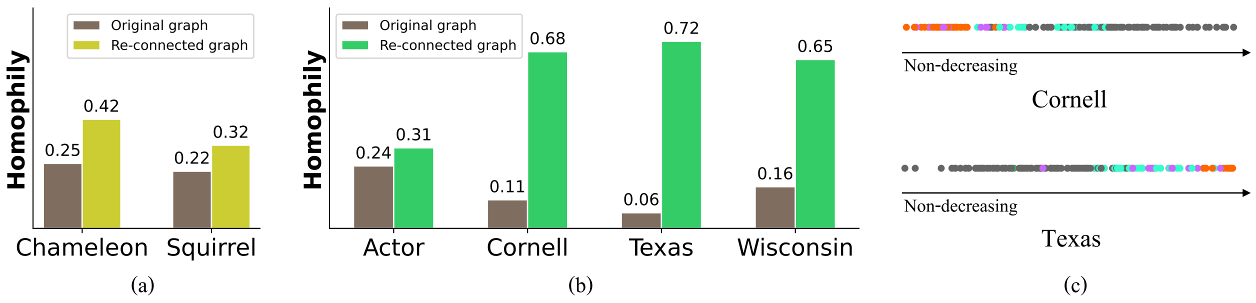


Figure 6.1: (a) Comparisons of the homophily between the original graph and the re-connected graph given by our NLGCN on *Chameleon* and *Squirrel*. (b) Comparisons of the homophily between the original graph and the re-connected graph given by our NLMLP on *Actor*, *Cornell*, *Texas*, and *Wisconsin*. (c) Visualization of sorted node sequence after the attention-guided sorting for *Cornell* and *Texas*. The colors denote node labels. Details are explained in Section 6.4.4.

field of $\hat{z}_i$ as directly connected to z_i, *i.e.* z_i's 1-hop neighborhood. The information within this new 1-hop neighborhood will be aggregated. If our non-local GNNs indeed perform non-local aggregation, the homophily of the re-connected graph should be larger than the original graph. Therefore, we compute $H(\hat{\mathcal{G}})$ for each dataset to verify this statement. Following Section 6.4.3, we apply NLMLP on *Actor*, *Cornell*, *Texas*, and *Wisconsin* and NLGCN on *Chameleon* and *Squirrel*.

Figure 6.1 compares $H(\hat{\mathcal{G}})$ with $H(\mathcal{G})$ for each dataset. We can observe that $H(\hat{\mathcal{G}})$ is much larger than $H(\mathcal{G})$, indicating that distant but informative nodes are near each other in the re-connected graph $\hat{\mathcal{G}}$. We also provide the visualizations of the sorted sequence for *Cornell* and *Texas*. We can see that nodes with the same label tend to be clustered together. These facts indicate that our non-local GNNs perform non-local aggregation with the attention-guided sorting.

6.4.5 Efficiency Comparisons

Table 6.6: Comparisons in terms of real running time (*milliseconds*).

	Chameleon	**Squirrel**
GCN	$22.2\ (1.0\times)$	$14.3\ (1.0\times)$
GAT	$33.2\ (1.5\times)$	$163.3\ (11.4\times)$
Geom-GCN	$3615.0\ (163.1\times)$	$10430.0\ (727.3\times)$
NLGCN	$26.3\ (1.2\times)$	$39.6\ (2.8\times)$

As analyzed in Section 6.3.2, our proposed non-local aggregation framework is more efficient than previous methods based on the original attention mechanism, such as Geom-GCN [173]. Concretely, our method requires only $O(n \log n)$ computation time in contrast to $O(n^2)$. In this section, we compare the real running time to verify our analysis. Specifically, we compare NLGCN with Geom-GCN as well as GCN and GAT. For Geom-GCN, we use the code provided in [173]. Each model is trained for 500 epochs on each dataset and the average training time per epoch is reported.

The results are shown in Table 6.6. Although our NLGCN is built upon GCN, it is just slightly slower than GCN and faster than GAT, showing the efficiency of our non-local aggregation framework. On the other hand, Geom-GCN is significantly slower due to the fact that it has $O(n^2)$ time complexity.

7. CONCLUSIONS AND FUTURE WORK

In Chapter 2, we propose the separable and shared (SS) output layer based on the attention mechanism, through our our exploration of solutions to the gridding problem in dilated convolutions. We start by proposing two simple yet effective degridding methods based on a decomposition of dilated convolutions. The proposed methods differ from existing degridding approaches in two aspects. First, we address the gridding artifacts in terms of a single dilated convolution operation instead of multiple layers in cascade. Second, our methods only require learning a negligible amount of extra parameters. Experimental results show that they improve DCNNs with dilated convolutions significantly and consistently. The smoothing effect is also visualized in the effective receptive field (ERF) analysis. Through further analysis, we relate both proposed methods together and define the SS operators. The newly defined SS operation is a general neural network operation and may result in a general degridding strategy. We explore this direction in this updated version and propose the SS output layer, which employs the attention mechanism is able to smooth the entire network by only replacing the output layer and obtain improved performance.

In Chapter 3, we propose the non-local U-Nets for biomedical image segmentation. As pointed out, prior U-Net based models do not have an efficient and effective way to aggregate global information by using stacked local operators only, which limits their performance. To address these problems, we propose a global aggregation block which can be flexibly used in U-Net for size-preserving, down-sampling and up-sampling processes. In particular, we are the first to extend the attention mechanism for down-sampling and up-sampling processes. Experiments on the 3D multimodality isointense infant brain MR image segmentation task show that, with global aggregation blocks, our non-local U-Nets outperform previous models significantly with fewer parameters and faster computation.

In Chapter 4, we introduce global voxel transformer networks (GVTNets) built upon attention-based global voxel transformer operators (GVTOs), an advanced deep learning tool for augmented microscopy. Compared to the U-Net, GVTNets are more powerful models that are capable of capturing long-range dependencies and selectively aggregating global information for inputs of any spatial size. With GVTNets, various augmented microscopy tasks can be performed with significantly improved accuracy, such as predicting the fluorescence images of subcellular structures directly from transmitted-light images without using fluorescent labels, conducting content-aware image denoising, and projecting a 3D microscope image to a 2D surface for analysis. We have demonstrated the superiority of GVTNets and GVTOs on several publicly available datasets for augmented microscopy [107, 109, 101]. In particular, we have provided examples where GVTNets achieve better inference performance with inputs of larger spatial sizes, indicating the ability of utilizing global information. In addition, besides the supervised learning setting, GVTNets outperform the U-Net under a simple transfer learning setting, showing better generalization ability due to input-dependent weights.

In Chapter 5, we propose to the attentional second-order pooling for graph neural networks (GNNs). In particular, we propose to perform graph representation learning with second-order pooling, by pointing out that second-order pooling can naturally solve the challenges of graph pooling. Second-order pooling is more powerful than existing graph pooling methods, since it is capable of using all node information and collecting second-order statistics that encode feature correlations and topology information. To take advantage of second-order pooling in graph representation learning, we propose two global graph pooling approaches based on second-order pooling; namely, bilinear mapping and attentional second-order pooling. Our proposed methods solve the practical problems incurred by directly using second-order pooling with GNNs. We theoretically show that our proposed methods are more suitable to graph representation learning by comparing with two related pooling methods from computer vision tasks. In addition, we extend one of the proposed method to a hierarchical graph pooling method, which has more flexibility. To demonstrate the effectiveness of our methods, we conduct thorough experiments on graph classification tasks. Our proposed methods have achieved the new state-of-the-art performance on eight out of nine benchmark datasets. Ablation studies are performed to show that our methods outperform existing graph pooling methods significantly and achieve good performance consistently with different GNNs.

In Chapter 6, we propose a simple yet effective non-local aggregation framework for GNNs and develop non-local GNNs. The core of the framework is an efficient attention-guided sorting, which enables non-local aggregation through convolution. The proposed framework can be easily used to build non-local GNNs with low computational costs. We perform thorough experiments on node classification tasks to evaluate our proposed method. In particular, we experimentally analyze existing disassortative graph datasets and apply different non-local GNNs accordingly. The results show that our non-local GNNs significantly outperform previous state-of-the-art methods on six benchmark datasets of disassortative graphs, in terms of both accuracy and speed.

In terms of future work, we discuss three possible directions. First, in our studies on deep attention networks (DANs) for images, we note that the attention mechanism itself is not computationally friendly for high dimensional data. Therefore, developing efficient variants of the attention mechanism for high-dimensional data is important. We have done some preliminary studies on it [185]. Second, the success of DANs in natural language processing is highly related to self-supervised learning and transfer learning. While we investigate the generalization ability of the attention mechanism under a simple transfer learning setting in Chapter 4, this direction has not been well explored. Third, our proposed DANs are not stand-alone attention networks. In particular, we use a combination of convolutions and the attention mechanism. On the other hand, there is no convolution or recurrence in successful DANs for texts. Exploring stand-alone attention networks for images and graphs would be an interesting direction, with some preliminary studies in the literature [186, 17].

REFERENCES

[1] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[2] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," in *Proceedings of Joint ICML and IJCAI Workshop on Computational Biology*, 2018.

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proceedings of the International Conference on Learning Representations*, 2019.

[4] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

[5] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 723–731, ACM, 2019.

[6] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the International Conference on Machine Learning*, pp. 3734–3743, 2019.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, IEEE, 2016.

[11] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

[15] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *Proceedings of the International Conference on Learning Representations*, 2017.

[16] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in *Proceedings of the International Conference on Learning Representations*, 2018.

[18] H. Gao, Z. Wang, and S. Ji, “Large-scale learnable graph convolutional networks,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1416–1424, ACM, 2018.

[19] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the International Conference on Machine Learning*, pp. 2048–2057, 2015.

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the International Conference on Learning Representations*, 2015.

[21] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems*, pp. 577–585, 2015.

[22] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-attention for visual question answering," in *Advances in Neural Information Processing Systems*, pp. 289–297, 2016.

[23] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 21–29, IEEE, 2016.

[24] L. Li, S. Tang, L. Deng, Y. Zhang, and Q. Tian, "Image caption with global-local attention," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[25] Z. Wang and S. Ji, "Learning convolutional text representations for visual question answering," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 594–602, SIAM, 2018.

[26] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489, 2016.

[27] Z. Wang and S. Ji, “Smoothed dilated convolutions for improved dense prediction,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2486–2495, ACM, 2018.

[28] Z. Wang, N. Zou, D. Shen, and S. Ji, “Non-local U-Nets for biomedical image segmentation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 6315–6322, 2020.

[29] Z. Wang, Y. Xie, and S. Ji, “Global voxel transformer networks for augmented microscopy,” *arXiv preprint arXiv:2008.02340*, 2020.

[30] Z. Wang and S. Ji, “Second-order pooling for graph neural networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[31] M. Liu, Z. Wang, and S. Ji, “Non-local graph neural networks,” *arXiv preprint arXiv:2005.14612*, 2020.

[32] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber, “Fast image scanning with deep max-pooling convolutional neural networks,” in *Proceedings of the IEEE International Conference on Image Processing*, pp. 4034–4038, IEEE, 2013.

[33] H. Li, R. Zhao, and X. Wang, “Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification,” *arXiv preprint arXiv:1412.4526*, 2014.

[34] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” in *Proceedings of the International Conference on Learning Representations*, 2016.

[35] F. Yu, V. Koltun, and T. Funkhouser, “Dilated residual networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 472–480, IEEE, 2017.

[36] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[37] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[38] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pp. 1451–1460, IEEE, 2018.

[39] R. Hamaguchi, A. Fujita, K. Nemoto, T. Imaizumi, and S. Hikosaka, "Effective use of dilated convolutions for segmenting small object instances in remote sensing imagery," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pp. 1442–1450, IEEE, 2018.

[40] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2881–2890, IEEE, 2017.

[41] H. Gao, H. Yuan, Z. Wang, and S. Ji, "Pixel transposed convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 5, pp. 1218–1227, 2019.

[42] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2014.

[43] G. Papandreou, I. Kokkinos, and P.-A. Savalle, "Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 390–399, IEEE, 2015.

[44] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in Neural Information Processing Systems*, pp. 379–387, 2016.

[45] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7310–7311, IEEE, 2017.

[46] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[47] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," in *Proceedings of the International Conference on Machine Learning*, pp. 1771–1779, 2017.

[48] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[49] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets*, pp. 286–297, Springer, 1990.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, IEEE, 2009.

[51] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, IEEE, 2015.

[52] M. J. Shensa, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *IEEE Transactions on Signal Processing*, vol. 40, no. 10, pp. 2464–2482, 1992.

[53] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceed-*

ings of the 12th {USENIX} Symposium on Operating Systems Design and Implementation, pp. 265–283, 2016.

[54] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 4898–4906, 2016.

[55] F. Mamalet and C. Garcia, "Simplifying convnets for fast learning," in *Proceedings of the International Conference on Artificial Neural Networks*, pp. 58–65, Springer, 2012.

[56] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1251–1258, IEEE, 2017.

[57] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5115–5124, IEEE, 2017.

[58] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[59] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3213–3223, IEEE, 2016.

[60] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 991–998, IEEE, 2011.

[61] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *arXiv preprint arXiv:1506.04579*, 2015.

[62] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the European Conference on Computer Vision*, pp. 740–755, Springer, 2014.

[63] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, Springer, 2015.

[64] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1925–1934, IEEE, 2017.

[65] A. Fakhry, T. Zeng, and S. Ji, "Residual deconvolutional networks for brain electron microscopy image segmentation," *IEEE Transactions on Medical Imaging*, vol. 36, no. 2, pp. 447–456, 2017.

[66] K. Lee, J. Zung, P. Li, V. Jain, and H. S. Seung, "Superhuman accuracy on the SNEMI3D connectomics challenge," *arXiv preprint arXiv:1706.00120*, 2017.

[67] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-net: learning dense volumetric segmentation from sparse annotation," in *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 424–432, Springer, 2016.

[68] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proceedings of the International Conference on 3D Vision*, pp. 565–571, IEEE, 2016.

[69] D. Nie, L. Wang, E. Adeli, C. Lao, W. Lin, and D. Shen, "3-D fully convolutional networks for multimodal isointense infant brain image segmentation," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 1123–1136, 2018.

[70] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, IEEE, 2018.

[71] H. Yuan, N. Zou, S. Zhang, H. Peng, and S. Ji, "Learning hierarchical and shared features for improving 3D neuron reconstruction," in *Proceedings of the IEEE International Conference on Data Mining*, pp. 806–815, IEEE, 2019.

[72] H. Yuan, L. Cai, Z. Wang, X. Hu, S. Zhang, and S. Ji, "Computational modeling of cellular structures using conditional deep generative networks," *Bioinformatics*, vol. 35, no. 12, pp. 2141–2149, 2018.

[73] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2117–2125, IEEE, 2017.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proceedings of the European Conference on Computer Vision*, pp. 630–645, Springer, 2016.

[75] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning*, pp. 448–456, 2015.

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[77] Y. Zhang, M. Brady, and S. Smith, "Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm," *IEEE Transactions on Medical Imaging*, vol. 20, no. 1, pp. 45–57, 2001.

[78] A. Criminisi and J. Shotton, *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

[79] L. Wang, Y. Gao, F. Shi, G. Li, J. H. Gilmore, W. Lin, and D. Shen, "Links: Learning-based multi-source integration framework for segmentation of infant brain images," *Neuroimage*, vol. 108, pp. 160–172, 2015.

[80] W. Zhang, R. Li, H. Deng, L. Wang, W. Lin, S. Ji, and D. Shen, "Deep convolutional neural networks for multi-modality isointense infant brain image segmentation," *Neuroimage*, vol. 108, pp. 214–224, 2015.

[81] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker, "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation," *Medical Image Analysis*, vol. 36, pp. 61–78, 2017.

[82] M.-P. Dubuisson and A. K. Jain, "A modified hausdorff distance for object matching," in *Proceedings of the International Conference on Pattern Recognition*, vol. 1, pp. 566–568, IEEE, 1994.

[83] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, pp. 950–957, 1992.

[84] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.

[85] M. G. Gustafsson, "Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy," *Journal of Microscopy*, vol. 198, no. 2, pp. 82–87, 2000.

[86] J. Huisken, J. Swoger, F. Del Bene, J. Wittbrodt, and E. H. Stelzer, "Optical sectioning deep inside live embryos by selective plane illumination microscopy," *Science*, vol. 305, no. 5686, pp. 1007–1009, 2004.

[87] E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, "Imaging intracellular fluorescent proteins at nanometer resolution," *Science*, vol. 313, no. 5793, pp. 1642–1645, 2006.

[88] M. J. Rust, M. Bates, and X. Zhuang, "Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (storm)," *Nature Methods*, vol. 3, no. 10, p. 793, 2006.

[89] R. Heintzmann and M. G. Gustafsson, “Subdiffraction resolution in continuous samples,” *Nature Photonics*, vol. 3, no. 7, p. 362, 2009.

[90] R. Tomer, K. Khairy, F. Amat, and P. J. Keller, “Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy,” *Nature Methods*, vol. 9, no. 7, p. 755, 2012.

[91] B.-C. Chen, W. R. Legant, K. Wang, L. Shao, D. E. Milkie, M. W. Davidson, C. Janetopoulos, X. S. Wu, J. A. Hammer, Z. Liu, *et al.*, “Lattice light-sheet microscopy: imaging molecules to embryos at high spatiotemporal resolution,” *Science*, vol. 346, no. 6208, p. 1257998, 2014.

[92] C. Belthangady and L. A. Royer, “Applications, promises, and pitfalls of deep learning for fluorescence image reconstruction,” *Nature Methods*, pp. 1–11, 2019.

[93] P. P. Laissue, R. A. Alghamdi, P. Tomancak, E. G. Reynaud, and H. Shroff, “Assessing phototoxicity in live fluorescence imaging,” *Nature Methods*, vol. 14, no. 7, p. 657, 2017.

[94] J. Icha, M. Weber, J. C. Waters, and C. Norden, “Phototoxicity in live fluorescence microscopy, and how to avoid it,” *BioEssays*, vol. 39, no. 8, p. 1700003, 2017.

[95] J. Selinummi, P. Ruusuvuori, I. Podolsky, A. Ozinsky, E. Gold, O. Yli-Harja, A. Aderem, and I. Shmulevich, “Bright field microscopy as an alternative to whole cell fluorescence in automated analysis of macrophage images,” *PloS One*, vol. 4, no. 10, p. e7497, 2009.

[96] J. B. Pawley, “Fundamental limits in confocal microscopy,” in *Handbook of Biological Confocal Microscopy*, pp. 20–42, Springer, 2006.

[97] N. Scherf and J. Huisken, “The smart and gentle microscope,” *Nature Biotechnology*, vol. 33, no. 8, p. 815, 2015.

[98] S. Skylaki, O. Hilsenbeck, and T. Schroeder, “Challenges in long-term imaging and quantification of single-cell dynamics,” *Nature Biotechnology*, vol. 34, no. 11, p. 1137, 2016.

[99] D. P. Sullivan and E. Lundberg, “Seeing more: a future of augmented microscopy,” *Cell*, vol. 173, no. 3, pp. 546–548, 2018.

[100] P. Chen, K. Gadepalli, R. MacDonald, Y. Liu, S. Kadowaki, K. Nagpal, T. Kohlberger, J. Dean, G. S. Corrado, J. D. Hipp, *et al.*, “An augmented reality microscope with real-time artificial intelligence integration for cancer diagnosis,” *Nature Medicine*, vol. 25, no. 9, pp. 1453–1457, 2019.

[101] E. Moen, D. Bannon, T. Kudo, W. Graf, M. Covert, and D. Van Valen, “Deep learning for cellular image analysis,” *Nature Methods*, pp. 1–14, 2019.

[102] G. R. Johnson, R. M. Donovan-Maiye, and M. M. Maleckar, “Building a 3D integrated cell,” *bioRxiv*, p. 238378, 2017.

[103] C. Ounkomol, D. A. Fernandes, S. Seshamani, M. M. Maleckar, F. Collman, and G. R. Johnson, “Three dimensional cross-modal image inference: label-free methods for subcellular structure prediction,” *bioRxiv*, p. 216606, 2017.

[104] A. Osokin, A. Chessel, R. E. Carazo Salas, and F. Vaggi, “GANs for biological image synthesis,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2233–2242, 2017.

[105] G. Johnson, R. Donovan-Maiye, C. Ounkomol, and M. M. Maleckar, “Studying stem cell organization using “label-free” methods and a novel generative adversarial model,” *Biophysical Journal*, vol. 114, no. 3, p. 43a, 2018.

[106] E. M. Christiansen, S. J. Yang, D. M. Ando, A. Javaherian, G. Skibinski, S. Lipnick, E. Mount, A. O’Neil, K. Shah, A. K. Lee, *et al.*, “In silico labeling: predicting fluorescent labels in unlabeled images,” *Cell*, vol. 173, no. 3, pp. 792–803, 2018.

[107] C. Ounkomol, S. Seshamani, M. M. Maleckar, F. Collman, and G. R. Johnson, “Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy,” *Nature Methods*, vol. 15, no. 11, p. 917, 2018.

[108] Y. Wu, Y. Rivenson, H. Wang, Y. Luo, E. Ben-David, L. A. Bentolila, C. Pritz, and A. Ozcan, “Three-dimensional virtual refocusing of fluorescence microscopy images using deep learning,” *Nature Methods*, pp. 1–9, 2019.

[109] M. Weigert, U. Schmidt, T. Boothe, A. Müller, A. Dibrov, A. Jain, B. Wilhelm, D. Schmidt, C. Broaddus, S. Culley, *et al.*, “Content-aware image restoration: pushing the limits of fluorescence microscopy,” *Nature Methods*, vol. 15, no. 12, p. 1090, 2018.

[110] H. Wang, Y. Rivenson, Y. Jin, Z. Wei, R. Gao, H. Gunaydin, L. Bentolila, and A. Ozcan, “Deep learning achieves super-resolution in fluorescence microscopy,” *Biorxiv*, p. 309641, 2018.

[111] H. Wang, Y. Rivenson, Y. Jin, Z. Wei, R. Gao, H. Günaydın, L. A. Bentolila, C. Kural, and A. Ozcan, “Deep learning enables cross-modality super-resolution in fluorescence microscopy,” *Nature Methods*, vol. 16, pp. 103–110, 2019.

[112] Y. Rivenson, Z. Göröcs, H. Günaydin, Y. Zhang, H. Wang, and A. Ozcan, “Deep learning microscopy,” *Optica*, vol. 4, no. 11, pp. 1437–1443, 2017.

[113] T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, K. Seiwald, *et al.*, “U-Net: deep learning for cell counting, detection, and morphometry,” *Nature Methods*, vol. 16, no. 1, p. 67, 2019.

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

[115] D. R. Wilson and T. R. Martinez, “The general inefficiency of batch training for gradient descent learning,” *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003.

[116] Q. Zhang, Z. Cui, X. Niu, S. Geng, and Y. Qiao, “Image segmentation with pyramid dilated convolution based on resnet and u-net,” in *Proceedings of the International Conference on Neural Information Processing*, pp. 364–372, Springer, 2017.

[117] J. Huang, P. Zhu, M. Geng, J. Ran, X. Zhou, C. Xing, P. Wan, and X. Ji, "Range scaling global u-net for perceptual image enhancement on mobile devices," in *Proceedings of the European Conference on Computer Vision Workshops*, vol. 2, Springer, 2018.

[118] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, *et al.*, "Attention U-Net: Learning where to look for the pancreas," *arXiv preprint arXiv:1804.03999*, 2018.

[119] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "UNet++: A nested u-net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pp. 3–11, Springer, 2018.

[120] Z. Gu, J. Cheng, H. Fu, K. Zhou, H. Hao, Y. Zhao, T. Zhang, S. Gao, and J. Liu, "Ce-net: context encoder network for 2d medical image segmentation," *IEEE Transactions on Medical Imaging*, vol. 38, no. 10, pp. 2281–2292, 2019.

[121] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[122] L. Cai, Z. Wang, H. Gao, D. Shen, and S. Ji, "Deep adversarial learning for multi-modality missing data completion," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1158–1166, ACM, 2018.

[123] Y. Rivenson, H. Wang, Z. Wei, K. de Haan, Y. Zhang, Y. Wu, H. Günaydın, J. E. Zuckerman, T. Chong, A. E. Sisk, *et al.*, "Virtual histological staining of unlabelled tissue-autofluorescence images via deep learning," *Nature Biomedical Engineering*, vol. 3, no. 6, p. 466, 2019.

[124] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[125] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," in *Advances in Neural Information Processing Systems*, pp. 5574–5584, 2017.

[126] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[127] B. Aigouy, R. Farhadifar, D. B. Staple, A. Sagner, J.-C. Röper, F. Jülicher, and S. Eaton, "Cell flow reorients the axis of planar polarity in the wing epithelium of drosophila," *Cell*, vol. 142, no. 5, pp. 773–786, 2010.

[128] R. Etournay, M. Popović, M. Merkel, A. Nandi, C. Blasse, B. Aigouy, H. Brandl, G. Myers, G. Salbreux, F. Jülicher, *et al.*, "Interplay of cell dynamics and epithelial tension during morphogenesis of the drosophila pupal wing," *Elife*, vol. 4, p. e07090, 2015.

[129] C. Blasse, S. Saalfeld, R. Etournay, A. Sagner, S. Eaton, and E. W. Myers, "Premosa: extracting 2d surfaces from 3d microscopy mosaics," *Bioinformatics*, vol. 33, no. 16, pp. 2563–2569, 2017.

[130] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[131] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1365–1374, ACM, 2015.

[132] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: Algorithms, applications and open challenges," in *Proceedings of the International Conference on Computational Social Networks*, pp. 79–91, Springer, 2018.

[133] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[134] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *Proceedings of the World Wide Web Conference*, pp. 417–426, ACM, 2019.

[135] H. Gao, J. Pei, and H. Huang, "Conditional random field enhanced graph convolutional neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 276–284, ACM, 2019.

[136] J. Ma, P. Cui, K. Kuang, X. Wang, and W. Zhu, "Disentangled graph convolutional networks," in *Proceedings of the International Conference on Machine Learning*, pp. 4212–4221, 2019.

[137] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *Proceedings of the World Wide Web Conference*, pp. 2022–2032, ACM, 2019.

[138] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proceedings of the International Conference on Machine Learning*, pp. 5449–5458, 2018.

[139] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," in *Advances in Neural Information Processing Systems*, pp. 991–1001, 2017.

[140] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

[141] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

[142] H. Gao and S. Ji, "Graph U-Nets," in *Proceedings of the International Conference on Machine Learning*, pp. 2083–2092, 2019.

[143] Y. Boureau, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in vision algorithms,” in *Proceedings of the International Conference on Machine learning*, vol. 345, 2010.

[144] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[145] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the International Conference on Computational Vision*, vol. 99, pp. 1150–1157, 1999.

[146] O. Tuzel, F. Porikli, and P. Meer, “Region covariance: A fast descriptor for detection and classification,” in *Proceedings of the European Conference on Computer Vision*, pp. 589–600, Springer, 2006.

[147] O. Tuzel, F. Porikli, and P. Meer, “Pedestrian detection via classification on riemannian manifolds,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1713–1727, 2008.

[148] F. Perronnin, J. Sánchez, and T. Mensink, “Improving the Fisher kernel for large-scale image classification,” in *Proceedings of the European Conference on Computer Vision*, pp. 143–156, Springer, 2010.

[149] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu, “Semantic segmentation with second-order pooling,” in *Proceedings of the European Conference on Computer Vision*, pp. 430–443, Springer, 2012.

[150] T.-Y. Lin, A. RoyChowdhury, and S. Maji, “Bilinear CNN models for fine-grained visual recognition,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1449–1457, 2015.

[151] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell, “Compact bilinear pooling,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 317–326, IEEE, 2016.

[152] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, “Multimodal compact bilinear pooling for visual question answering and visual grounding,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 457–468, 2016.

[153] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, “Geometric means in a novel vector space structure on symmetric positive-definite matrices,” *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 1, pp. 328–347, 2007.

[154] D. Acharya, Z. Huang, D. Pani Paudel, and L. Van Gool, “Covariance pooling for facial expression recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 367–374, 2018.

[155] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li, “Deep cnns meet global covariance pooling: Better representation and generalization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[156] R. Girdhar and D. Ramanan, “Attentional pooling for action recognition,” in *Advances in Neural Information Processing Systems*, pp. 34–45, 2017.

[157] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity,” *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[158] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, “Statistical evaluation of the predictive toxicology challenge 2000–2001,” *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[159] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[160] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[161] A. Shrivastava and P. Li, "A new space for comparing graphs," in *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 62–71, IEEE Press, 2014.

[162] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 177–187, ACM, 2005.

[163] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783, 2003.

[164] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the International Conference on Machine Learning*, pp. 2014–2023, 2016.

[165] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

[166] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[167] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[168] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," in *Proceedings of the International Conference on Machine Learning*, pp. 2191–2200, 2018.

[169] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1993–2001, 2016.

[170] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3693–3702, IEEE, 2017.

[171] H. Yuan and S. Ji, "StructPool: Structured graph pooling via conditional random fields," in *Proceedings of the International Conference on Learning Representations*, 2020.

[172] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proceedings of the International Conference on Learning Representations*, 2020.

[173] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2020.

[174] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[175] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," in *Advances in Neural Information Processing Systems*, pp. 6338–6347, 2017.

[176] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 385–394, ACM, 2017.

[177] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[178] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 807–816, ACM, 2009.

[179] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *arXiv preprint arXiv:1909.13021*, 2019.

[180] M. E. Newman, “Assortative mixing in networks,” *Physical review letters*, vol. 89, no. 20, p. 208701, 2002.

[181] H. Gao and S. Ji, “Graph representation learning via hard and channel-wise attention networks,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 741–749, ACM, 2019.

[182] B. Knyazev, G. W. Taylor, and M. Amer, “Understanding attention and generalization in graph neural networks,” in *Advances in Neural Information Processing Systems*, pp. 4204–4214, 2019.

[183] P. Adam, G. Sam, C. Soumith, C. Gregory, Y. Edward, D. Zachary, L. Zeming, D. Alban, A. Luca, and L. Adam, “Automatic differentiation in pytorch,” in *Proceedings of Neural Information Processing Systems Autodiff Workshop*, 2017.

[184] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” in *Proceedings of ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[185] H. Gao, Z. Wang, and S. Ji, “Kronecker attention networks,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 229–237, ACM, 2020.

[186] N. Parmar, P. Ramachandran, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, “Stand-alone self-attention in vision models,” in *Advances in Neural Information Processing Systems*, pp. 68–80, 2019.