Toward Efficient Visual Segmentation and Grouping

CHEN, Wanli

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

February 2024

Thesis Assessment Committee

Professor YOUNG Fung Yu (Chair)

Professor YU Bei (Thesis Supervisor)

Professor SHAO Zili (Thesis Co-supervisor)

Professor DOU Qi (Committee Member)

Professor SHI Hongjian (External Examiner)

Abstract

Visual segmentation and data grouping is comprehensively needed in the real-world. However, a good visual segmentation model should fully utilize all the information of the input data like image or point cloud, which requires large computational resources. In this research, we will handling the problem of computational cost in segmentation tasks and trying to propose an efficient visual segmentation system. We start our research at constructing efficient attention module in visual segmentation framework, then we explore the possibility of applying space-filling curve to visual segmentation. Finally, we successfully integrate the efficient attention module and space-filling curve to the visual segmentation system, producing an accurate, fast and memory-saving segmentation model. This thesis divides my research work into three parts, which will be introduced in the following paragraph.

In the first part of our research, we are focusing on optimizing attention block, which is the key component to perform high-accuracy segmentation. Attention block collects global context information of input features, which means it has no limited receptive field as convolution. However, this module is very costly and will compress the features in channel when generating spatial attention. Similarly, spatial features will also be compressed during channel attention generation. To handle the two problem, we propose a new attention generation scheme that based on tensor canonical-polyadic reconstruction. Specifically, we directly construct

3D attention map that has high rank by using a series of low-rank tensors. Our framework is named as RecoNet, which not only outperforms previous attention based methods in various dataset but also produces lower computational cost. Our proposed method is $100\times$ faster than previou works.

In the second part of our research, we focus on the application of space-filling curves (SFCs) in data grouping. It is a new approach to perform efficient visual segmentation as we can use SFCs to perform data clustering and compression first and then input the data into the visual segmentation system to lower its computational overhead. In this part, we first analyze the properties of different space-filling curves and then propose our SFC generation scheme. Specifically, we construct a deep-learning-based SFC generation framework coupled with our proposed efficient-GCN and Siamese Network learning scheme. The evaluation results show that the SFC generated by our method has better clustering properties compared with traditional SFCs.

In the third part, we combine low-rank attention with the space-filling curve, producing an efficient point cloud segmentation system called HilbertNet. We use 3D voxel data as the input, which has better spatial locality compared with raw point cloud, but it also occupies more computational resources. Here we introduce the Hilbert curve to compress 3D voxel data into 2D image data first, which greatly reduces the computational cost. Then we use our proposed low-rank attention module called Hilbert attention to extract point cloud features with high efficiency. We also propose Hilbert interpolation and Hilbert pooling modules to accommodate the distribution of compressed 2D data. HilbertNet shows top performance on several mainstream datasets while maintaining low cost and fast inference speed.

摘要

視覺分割和數據分組在現實世界中有著廣泛的需求。然而，一個好的視覺分割模型應該充分利用輸入數據如圖像或點雲的所有上下文信息，這通常需要大量的計算資源。在這項研究中，我們將直面并解決分割任務中的計算成本問題，並嘗試提出一個高效的視覺分割系統。我們的研究從構建視覺分割框架中的高效注意力模塊開始，然後我們探索將空間填充曲線應用於視覺分割的可能性。最後，我們成功地將高效的注意力模塊和空間填充曲線整合到視覺分割系統中，搭建了一個準確、快速且節省GPU存的分割模型。本文我將我的研究工作分為三部分，并且將在後續段落中詳細介紹。

在我們研究的第一部分，我們專注於優化注意力模塊，這是執行高精度分割的關鍵組件。注意力模塊可以收集輸入特徵的全局上下文信息，這意味著它沒有像卷積那樣有受限的感受野。然而，這個模塊通常非常消耗計算資源。同時，在生成空間注意力時它會壓縮通道的特徵。同樣，在通道注意力生成過程中，空間特徵也會被壓縮。為了處理上述兩個問題，我們提出了一種基於張量CP重構的新的注意力生成方案。具體來說，我們直接使用一系列低秩張量構建具有高秩特徵的3D注意力圖。我們的視覺分割框架名為RecoNet，它不僅在各種數據集中的表現超越了以前基於注意力的方法，而且需要的計算成本更低。具體來，我們提出的方法比以前的工作快100×。

在我們研究的第二部分，我們專注於空間填充曲線（SFCs）在數據分組中的應用。這是一種新的方法來實現高效的視覺分割，因為我們可以先使用SFCs進

行數據聚類和壓縮，然後再將數據輸入視覺分割系統以降低其計算開銷。在這部分，我們首先分析了不同空間填充曲線的屬性，然後提出了我們的SFC生成方案。具體來說，我們構建了一個基於深度學習的SFC生成框架，並結合了我們提出的efficient-GCN和連體網絡學習方案。評估結果顯示，我們方法生成的SFC與傳統SFC相比具有更好的聚類特性。

在第三部分，我們將低秩注意力與空間填充曲線結合起來，創建了一個稱為HilbertNet的高效點雲分割系統。我們使用3D體素數據作為輸入，它與原始點雲相比具有更好的空間局部性，但它也佔用了更多的計算資源。在這裡，我們首先引入希爾伯特曲線來將3D體素數據壓縮成2D圖像數據，這大大降低了後續的計算成本。然後我們使用我們提出的低秩注意力模塊，稱為Hilbert attention，用以高效率提取點雲特徵。我們還提出了Hilbert插值和Hilbert池化模塊來適應被壓縮后2D數據的分佈。HilbertNet在幾個主流數據集上展現出了頂尖性能，同時保持了低計算資源的占用。

Acknowledgments

I spent an unforgettable four years pursuing my PhD at the Chinese University of Hong Kong. I am grateful to have had many companions on my journey to acquire knowledge.

I am deeply thankful to my advisor, Prof Bei Yu. Prof Yu has been a mentor to me, insisting on recruiting me into his group during my CUHK CSE PhD interview despite opposition from other interviewers. In terms of research, Prof Yu did not limit my areas of interest, but encouraged me to engage in the research I enjoyed, while arranging for me to intern in the industry as much as possible. During my gap year before enrollment, Prof Yu arranged for me to join Tencent Youtu Lab, a top-tier computer vision lab in China, where I achieved my first top conference paper of my doctoral career.

In the early stages of my PhD, I was a very introverted and silent person, reluctant to communicate with others. Prof Yu hoped that my internship at SmartMore Technologies would enhance my communication skills. Serving as an interviewer at SmartMore for a long time, the constant interaction with others brought me joy and I found a significant improvement in my expressive abilities.

Starting from September 2022, I entered the darkest period of my life. Diagnosed with anxiety due to confusion about my post-graduation prospects and concerns about the number of my papers, I began to suffer from long-term insomnia, palpitations, headaches, and memory loss. Prof Yu continued to support my research during my most difficult times, taking care of my emotions, not challenging or pushing me, and helping me avoid some trivial work, such as TA duties. The work I did during that time was eventually accepted by the International Conference on Computer Vision (ICCV), which was my first ICCV paper and the highest-ranked

academic conference I have been accepted to.

My life is gradually getting back on track, and I am deeply grateful to have such a caring advisor during my PhD journey. All the previous students I know who were diagnosed with similar symptoms eventually chose to quit without exception, but I successfully made it to the final step, thanks to the support of Prof Yu.

Additionally, I would like to extend special thanks to Guojin CHEN, who is not only my junior in the PhD program, but also a lifelong friend. Guojin initially joined our lab as a master's student, but he demonstrated exceptional coding skills, strong execution abilities, and a keen intuition for research. I still remember the all-nighter we pulled when submitting our first paper, and the sense of relief we felt as we rode the bus home after the submission was complete the next day. That feeling is unforgettable. Guojin later successfully joined our research group as a PhD student and published several top conference papers in a short period of time, achieving accomplishments beyond my imagination. Throughout my academic journey, Guojin has been an important partner in my research, playing a significant role in many of my publications. After I was diagnosed with anxiety, Guojin was the only friend to whom I could reveal my true feelings, as he could fully understand my experience. I hope our friendship lasts forever.

Contents

Chapter 1

Introduction

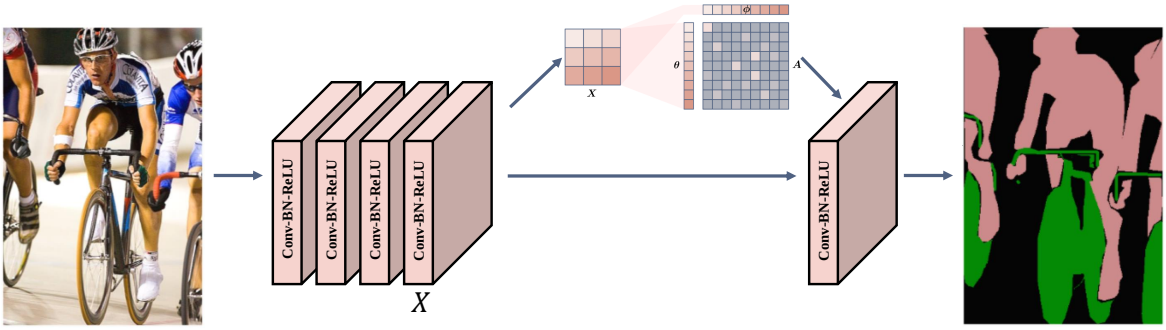


Figure 1.1: *The pipeline of a semantic segmentation algorithm begins with feeding the image into a deep neural network that includes a series of Conv-BN-ReLU layers. Then, the output feature X is processed by an attention block, which generates an attention map $\mathcal{A}$ using the linear mappings of X, marked as ϕ and θ. After that, the attention map is applied to help generate the final pixel-wise prediction.*

In recent years, the field of computer vision has witnessed remarkable advancements, largely due to the integration of deep neural networks. These networks, as they grow in depth and width, results in a significant increase in computational demands. Currently, the majority of state-of-the-art computer vision models achieve high-speed performance primarily with the assistance of GPUs. Nonetheless, numerous application scenarios are constrained by the availability of limited computational resources, such as those found in embedded systems and cellphone applications.

Consequently, there is a great need for the development of efficient deep learning algorithms that are capable of adapting to these resource-constrained platforms. Among the various tasks in computer vision, segmentation stands out as particularly computational heavy. The process of a segmentation algorithm is detailed in Figure 1.1. This process requires the classification of every pixel within an image, which means that the neural network designed for segmentation should be larger than those employed for other tasks in order to preserve the receptive fields, yet it simultaneously complicates the task of accelerating segmentation algorithms. The development of an efficient segmentation algorithm is crucial, as it forms the backbone of numerous real-world applications, including autonomous driving and image editing. Moreover, it is also widely used in other areas of computer vision, such as image generation and detection. In this thesis, we are focusing on how to accelerate the image/point cloud segmentation algorithm. We will first introduce how to perform efficient semantic segmentation, then we will propose how to achieve efficient data grouping using space-filling curves. Finally, we will show how to use space-filling curves to accelerate the segmentation task.

1.1 Efficient Semantic Segmentation

Semantic segmentation is a challenging task as it involves pixel-level classification for a given image. It requires the prediction model to recognize the shape, texture, and category of all pixels in the image. Fully Convolutional Networks (FCN) [68] were the first to apply deep neural networks to this task, making great progress in this field of study.

Recently, semantic segmentation has made significant strides by modeling context information in convolutional networks [89, 2, 59, 8, 131, 9]. The attention mechanism

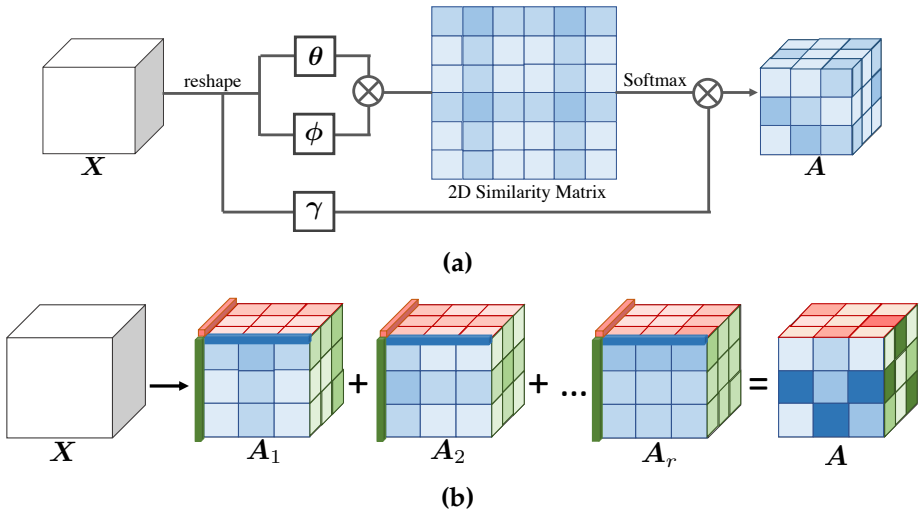


Figure 1.2: *(a) Vanilla non-local attention. (b) Attention low-rank reconstruction scheme. In vanilla non-local block, it requires 2D similarity matrix, which will cause the compression of channel information while in the attention low-rank reconstruction scheme, the attention map is constructed by 3D tensors which will avoid this problem.*

has become the most popular tool for constructing context features [132, 126, 31].

These approaches measure the importance of each element in a contextual tensor and produce an attention map which can identify the important part of the contextual tensor. The accurate context representation is then modeled through the attention map. However, these methods will inevitably compress the channel information during spatial attention modeling. Specifically, considering an attention map $\boldsymbol{A} \in \mathbb{R}^{HW \times HW}$, which is generated by the matrix multiplication of two inputs that have shapes of $HW \times C$ and $C \times HW$. After the multiplication, we can find that channel information in the C dimension is eliminated. As a result, the attention-based approaches will have well-represented spatial context features, but the channel context features will be ignored.

Intuitively, constructing context features directly without using a 2D attention map can bypass this problem. However, context features inherently have a high-rank nature [126] as they should possess enough information capacity to describe the

contextual information in an image. Additionally, the contextual information varies across different images. Therefore, context features cannot be effectively modeled by a very limited number of low-rank tensors. Due to this high-rank complexity, directly modeling context features presents a significant challenge.

In this research, we propose a novel method to model high-rank context features accurately and efficiently. According to the theory of tensor CP decomposition [46], we can progressively reconstruct the context features using a series of rank-1 tensors. Our approach can obtain both spatial and channel context information, which is different from previous approaches that had to compress either spatial or channel features. Figure 1.2 illustrates the difference between non-local attention and our proposed RecoNet attention mechanism.

Specifically, our RecoNet has two components: a tensor generation module (TGM) that generates rank-1 tensors, and a tensor reconstruction module (TRM) that reconstructs rank-1 tensors into a fine-grained high-rank tensor. TGM collects context information along the width, channel, and height directions, formulating them into a series of rank-1 tensors. Then, TRM reconstructs these low-rank tensors into a high-rank attention map following the workflow of CP reconstruction. This process identifies the occurrence of context information in different feature dimensions, and the high-rank context feature is then obtained with high accuracy and efficiency.

To measure the performance of our method, we conduct experiments on five popular datasets: PASCAL-VOC12, PASCAL-Context, ADE20K, COCO-Stuff, and SIFT-FLOW. Our approach consistently achieves state-of-the-art performance on these datasets, especially on PASCAL-VOC12 [23], where we ranked as **top-1** on the leaderboard. Additionally, since the attention map in RecoNet is constructed by low-rank tensors, the computational cost of our method is very small. For example,

we are **100** $\times$ faster than non-local attention.

However, continuous optimization of non-local attention does not always lead to a corresponding continuous improvement in performance. This optimization approach can easily reach a bottleneck, as a tensor rank that is too small cannot express all the information contained in an image. In pursuit of a more cost-effective way to carry information, we consider data linearization. Specifically, we aim to compress N-Dimensional matrices (like 2D or 3D) into 1D through linearization, allowing us to express rich information with a lower rank, thereby breaking through the performance bottleneck. Therefore, we need to consider how to efficiently perform data linearization. In this context, we propose the use of a space-filling curve for data linearization.

1.2 Efficient Data Grouping by SFCs

Space-filling curves (SFCs) can be used to linearize data, which means they can transform n-D data into a 1D sequence. This property has led to their extensive use in computer vision, such as data clustering [74], image compression [56], point cloud compression [103], geographic hashing [3], and data transmission [70]. They have also appeared in deep learning research, such as knowledge distillation [86] and point cloud analysis [10]. Traditional SFCs are formulated as fractal functions, such as the Hilbert curve [33] and the Z-curve [75]. Different SFCs have different properties, which can be used in different applications. For example, the Hilbert curve is locality-preserving, which makes it suitable for data compression, while the Z-curve has jump connections, which is feasible for faster data queries and can be used in database applications.

However, traditional SFCs usually have a fixed structure, which limits their

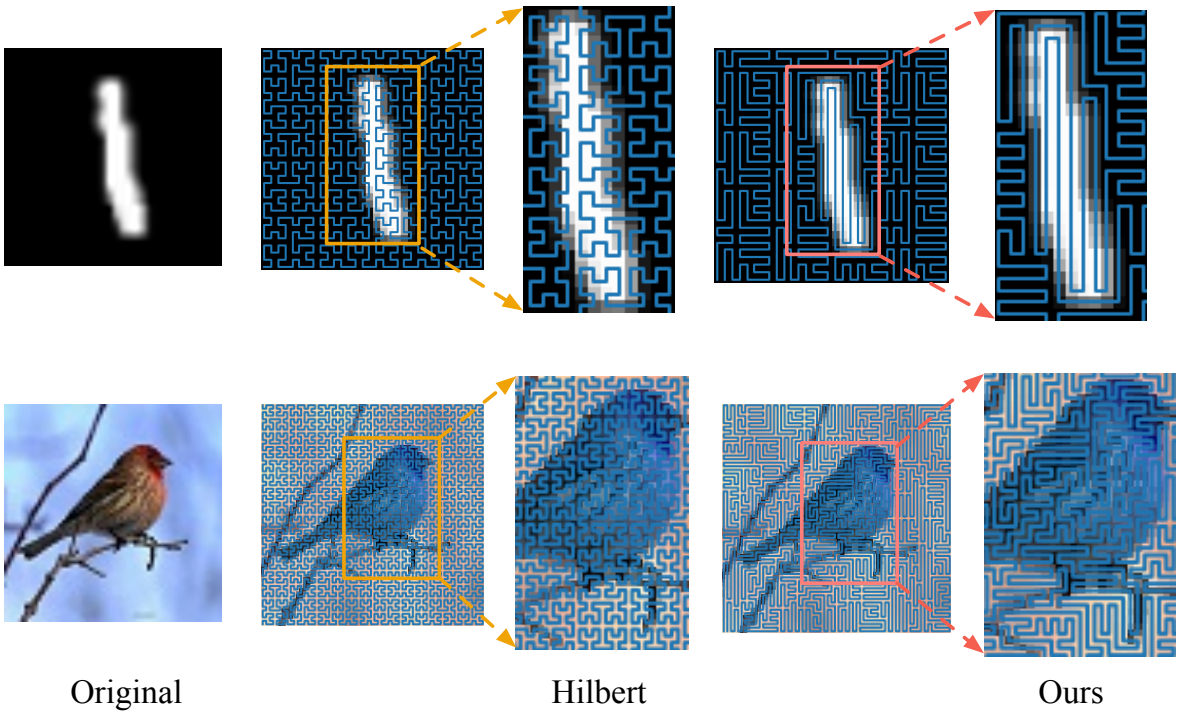


Figure 1.3: *The shape of Hilbert curve and a data-adaptive SFC. It can be found that the structure of Hilbert curve is fixed while in data-adaptive SFC, the curve varies according to the image content.*

applications. For example, we should use the Hilbert curve in data compression tasks instead of the Z-curve since it has jump connections, and the data flattened by the Z-curve will have a completely different data distribution compared to its original shape, which is harmful for data compression.

To obtain a versatile SFC, the space-filling curves should be data-adaptive (see Figure 1.3). In other words, the SFC should be generated according to the distribution of the given data so that we can use the obtained SFC in different tasks. Previous data-adaptive SFCs [139, 76, 17] mainly focus on image context. These SFCs are generated according to edges, image gradients, etc., providing better spatial locality compared with traditional SFCs. These approaches transform the SFC generation into a Hamiltonian path-finding problem, as illustrated in Figure 2.1. Recently, researchers have been trying to harness deep neural networks to generate more

powerful data-adaptive SFCs [102]. The SFCs in these works are generated by first constructing many small circuits on the given grid graph $\mathcal{G}$. Secondly, the dual graph $\mathcal{G}'$ is constructed based on $\mathcal{G}$, and the edge weights of $\mathcal{G}'$ are determined by the prediction of a GNN [43, 101]. After that, a minimum spanning tree searching algorithm like the Prim algorithm is applied to $\mathcal{G}'$, generating an MST. Finally, we can obtain a Hamiltonian circuit based on the given MST, and the Hamiltonian circuit itself will serve as an SFC. With the help of deep learning, the learning-based data-adaptive space-filling curves greatly outperform traditional SFCs. However, the large computational overhead and enormous GPU memory occupation limit their application. For example, given an image with a size of 64×64, its adjacency matrix will have the size of 4096×4096, which is not acceptable for efficient inference. Additionally, MST generation algorithms like Prim's [79] perform a greedy search, which is undifferentiable and cannot be optimized by gradient-based optimizers like SGD and Adam, which are widely used in neural networks.

In this research, we propose an efficient SFC generation method based on deep learning, which can solve the problem mentioned above. To handle the GNN scalability problem, we propose the Efficient-GCN (EGCN) module, which is specially designed for grid graphs. Our solution greatly reduces the computational cost of GNN, making it scalable to large grid graphs. Then we design a novel learning scheme based on the Siamese network to handle the network optimization problem and make the entire framework end-to-end trainable. Our proposed model works as follows:

Firstly, the input images are encoded by convolutional networks [32] and then normalized by a feature normalization module. After that, the output feature is fed into the EGCN module for efficient GCN calculation. The EGCN accelerates the GCN algorithm by fully utilizing the adjacency matrix structure of the grid

graph. Specifically, we observe that the grid graph only has 4 connections for each node, which means only 4 offset-diagonals appear in their adjacency matrices. Since the multiplication of a matrix A and an offset-diagonal matrix B is the same as shifting and padding matrix A, we do not need to perform *real* matrix multiplication between A and B; instead, we can achieve the same result by shifting A. EGCN achieves lower computational cost based on this observation.

After the calculation of EGCN, our proposed network optimization scheme is applied. Note that the generation of SFC is the problem of finding a Hamiltonian path, which is not differentiable, and the solution to combinatorial optimization should be applied in this scenario, such as Reinforcement Learning [40, 19, 47]. However, previous work [102] pointed out that RL is not stable in the curve generation task. Therefore, we need to design a new optimization scheme. Based on the self-learning technique [28, 11], we use the output of a Siamese network to supervise the main network, which indirectly optimizes the entire framework.

Furthermore, we propose multi-stage MST, which combines the minimum spanning tree of different convolution layers such that the generated SFC is more stable and has better performance.

After introducing the applications and functions of space-filling curves, we will provide an example to demonstrate how to apply them to visual segmentation.

1.3 Applying SFCs in Point Cloud Segmentation

Point clouds are the primary data format in 3D space and they are generated by 3D sensors like LiDAR. In the early stages, researchers conducted point cloud analysis using point-based methods, which only utilize point cloud data such as SO-Net [52], PointNet++ [85], and PointNet [82]. These approaches are fast and

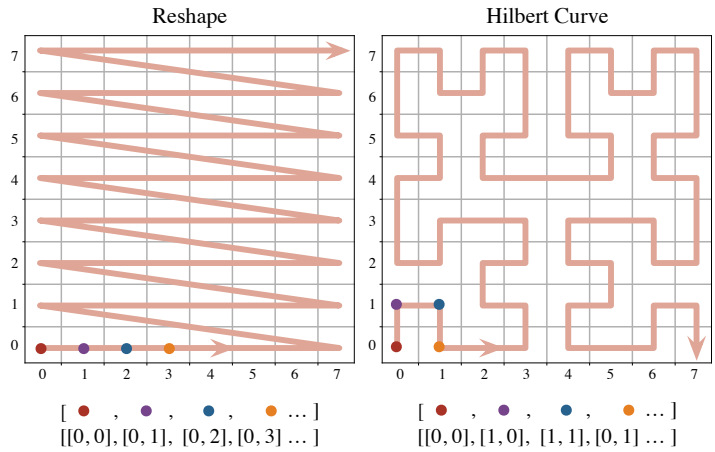


Figure 1.4: *Left: "Reshape" function or zigzag curve. Right: Hilbert curve. We can easily find that the "reshape" function has many "jump connections," while the Hilbert curve does not, which implies that the Hilbert curve has a better locality-preserving property.*

low-cost since they avoid using computationally heavy components such as 3D voxels and 2D meshes. However, point cloud data lacks spatial locality, which limits the performance of such methods. Recently, with the rapid development of convolutional neural networks, researchers are trying to use convolution in point cloud analysis since convolution is capable of handling spatial locality. A common pipeline for using convolution in point cloud analysis is to first transfer the original point cloud into a voxel. After that, 3D convolutions are applied to extract volumetric features. By using 3D convolution, the spatial locality of the point cloud will be well preserved. However, 3D convolution will occupy a large amount of GPU memory and consume a lot of computational resources.

An intuitive way to solve this problem is to find a mapping function from 3D space to 2D space, like bird-eye view images, flattening, multi-view images, and range images. Then we can use convolutional networks to process the 2D data. However, such mapping functions will change the data distribution of the original input point cloud and weaken its locality. Therefore, we should find a 3D to 2D

mapping that preserves the 3D locality.

The application of Space-filling curves [73] may meet this need. They are **fractal** functions that have been extensively applied in many tasks such as image compression [56], databases [3], and GIS [92]. The SFCs act as a single line that links all the elements in an N-dimensional space together without any repetition. This characteristic proposes a novel approach to reduce the dimension of a high-dimensional feature. For example, we can reduce the dimension of an image from 2D to a 1D sequence according to the mapping rule of a given SFC. There are various space-filling curves like the Hilbert curve [33], Z-curve [75], and Gray-Code [24], each with a different mapping rule. In this research, we will use the Hilbert curve as shown in Figure 1.4. Since the mapping rule of the Hilbert curve has a good locality-preserving property, the sequence that is flattened by the Hilbert curve will preserve the spatial information of its original shape.

Specifically, we first convert the point cloud into voxels, as voxel data has richer locality information compared to point cloud data. Then, we flatten the voxels slice-by-slice along the Z-axis as mentioned in Equation (5.3), thereby obtaining a 2D representation of the 3D voxels. During the flattening process, each slice is transformed into 1D data via the Hilbert curve. The flattened data retains the locality of its original shape, as the Hilbert curve is locality-preserving.

It is important to note that the data distribution of a flattened voxel differs from that of a traditional 2D image. For instance, adjacent pixels in the flattened voxel may not be neighbors in its original 3D shape. Therefore, we have specifically designed a pooling module called Hilbert pooling and an attention module named Hilbert attention to extract features in the flattened voxel. Additionally, we propose Hilbert interpolation to compress 2D features and merge them with the output of the 1D branch. We refer to the combination of these operations as HilbertNet.

To evaluate the performance of HilbertNet, we conducted several experiments, including point cloud segmentation and point cloud classification. The experimental results demonstrate that our proposed framework achieves top performance on the ModelNet40 [112], ShapeNet [7], and S3DIS [1] datasets, while maintaining small computational costs.

Chapter 2

Literature Review

2.1 Efficient Semantic Segmentation

Tensor Low-rank Representation. According to the theory of tensor decomposition [46], a high-rank tensor can be decomposed into many low-rank tensors. In other words, a high-rank tensor can be reconstructed through the linear combination of several low-rank tensors. This property of tensors is widely applied in various computer vision tasks. For instance, researchers have decomposed the weight tensor in convolutions to accelerate inference [50], and it has also been utilized for model compression [124]. There are two commonly used tensor decomposition methods: CP decomposition and Tucker decomposition. During Tucker decomposition, the original tensor is separated into a core tensor and several matrices, whereas CP decomposition transforms the high-rank tensor into a series of vectors (rank-1 tensors). In this thesis, our focus is on tensor *reconstruction* rather than decomposition. Specifically, we employ CP reconstruction to rebuild a high-rank attention map from multiple rank-1 tensors, thereby enabling efficient computation.

Self-Attention in Computer Vision. Researchers in natural language processing

(NLP) first proposed the attention mechanism [100, 13, 121, 16], which is a method for encoding long-range features. This property of attention makes it popular not only in NLP but also in computer vision tasks, as the receptive field of attention is not limited, unlike convolution which confines its receptive field within the convolution kernel. For example, SE-Net [37] introduced channel-wise attention to boost the performance of ResNet in image classification tasks. CBAM [108] further improved the ResNet model by integrating both spatial and channel attention into the original CNN model. To enlarge the receptive field of neural networks, Wang *et al*. invented the non-local network [106], which is applicable not only to images but also to video applications.

Context Aggregation in Semantic Segmentation. The success of semantic segmentation is largely attributed to the collection of contextual information. The richer the contextual information we gather, the better the segmentation results we achieve. Contextual information can be collected by enlarging the receptive field of the neural network. For instance, FCN [68] merges the output features of different layers in the neural network to enrich the contextual information. Subsequently, feature pyramid approaches like DeepLab [8, 9] and PSPNet [131] were proposed to gather contextual information by applying large convolution kernels. However, these methods cannot differentiate the importance of the contextual features they collect. To address this issue, attention-based approaches like PSANet [132], APCNet [31], CFNet [126], and EncNet [125] were introduced. Although attention-based methods collect contextual information efficiently, they consume a significant amount of computational resources. Consequently, efficient attention methods such as CCNet [38], EMANet [54], and A^2Net [12] were proposed. These methods simplified the attention generation algorithm, making semantic segmentation both accurate and efficient. However, these methods can only generate attention in either the spatial

or channel direction, meaning they have to sacrifice either the channel context or spatial context. Unlike these previous approaches, our proposed method constructs attention directly from a rank-1 tensor, which means we can obtain both spatial and channel attention simultaneously.

2.2 Efficient Data Grouping via SFCs

Fractal Space-Filling Curves. Space-filling curves are surjective mappings from 1D space $\mathbb{R}$ to N-D space $\mathbb{R}^N$. In other words, they can convert data from N-D space to 1D. Given a unit function $f_1(x)$, we can create a fractal space-filling curve by repeating it infinitely. The first fractal space-filling curve was proposed by Peano in 1890, which is called the Peano curve [91]. It maps the data in [0, 1] to $[0,1]^2$ and has the analytical formulation $f(x) = \lim_{n\to\infty} f_n(x)$. The detailed expression of the Peano curve, which is composed of a parametric equation $t \to (\varphi(t), \psi(t))$, is given by Cesàro [91]:

$$\varphi(t) = \sum_{n=1}^{\infty} \frac{f_{2n-1}(t)}{3^n}, \quad \psi(t) = \sum_{n=1}^{\infty} \frac{f_{2n}(t)}{3^n},$$

where

$$f_m(t) = 1 + ([3^m t] - 3[3^{m-1} t] - 1)(-1)^{[3t]+[3^2 t]+\ldots+[3^{m-1} t]}.$$

If we simply change the parametric equation $t \to (\varphi(t), \psi(t))$ by replacing $f(t)$ with $f(t) = 1 - t, t \in [0, 1]$, we can get a variant of Peano curve, which is called Z-curve and was proposed in 1904 by Lebesgue [75]:

$$\varphi(t) = \sum_{n=1}^{\infty} \frac{f\left(3^{2n-2} t\right)}{2^n}, \quad \psi(t) = \sum_{n=1}^{\infty} \frac{f\left(3^{2n-1} t\right)}{2^n},$$

Z-curve has many Z-shaped connections, which can connect two points with large distance and therefore, such space-filling curve is welcomed in the tasks that requires

low latency query like data indexing [103] and geographic hashing [3]. We can also generate space filling curves that has no jump connections such as Hilbert curve [74]. To generate Hilbert curve, we should replace $f(t)$ to $f_h(x) = f_h\left(0_4^{\cdot} q_1 q_2 q_3 \ldots\right)$, where $f_h(x)$ is expressed as the quaternary Cantor set [91]. Therefore, the analytical formulation of Hilbert curve is:

$$f_h(x) = \sum_{j=1}^{\infty} \left(1/2^j\right) (-1)^{e_{0j}} \operatorname{sgn}\left(q_j\right) \begin{pmatrix} \left(1-d_j\right) q_j - 1 \\ 1 - d_j q_j \end{pmatrix},$$

$$\begin{aligned} e_{kj} &= \text{ number of k's preceding } q_j \text{ (mod 2)} \\ d_j &= e_{0j} + e_{3j} (mod 2) \end{aligned}$$

Hilbert curve is locality preserving since it has no jump connection. Hence, it is widely used in the tasks that requires feature invariant such as point cloud compression [103] and image compression.

Different fractal space-filling curves have different properties and can be used under certain conditions. Their advantage is that they can be generated with low cost. However, the shape of fractal space-filling curves is predefined and cannot adapt to the distribution of data, which limits their application. To make the SFC more versatile, researchers proposed data-adaptive SFC. The shape of the curve in data-adaptive SFC varies according to the distribution of given data, and thus they can be applied to different tasks without changing the curve generation method.

Data-adaptive Space-Filling Curves. Data-adaptive space-filling curves are usually generated according to the gradient [76], image context [17], and neighborhood similarity [139], which makes them adaptive to such features and therefore can be applied to various situations with better performance compared with fractal SFCs.

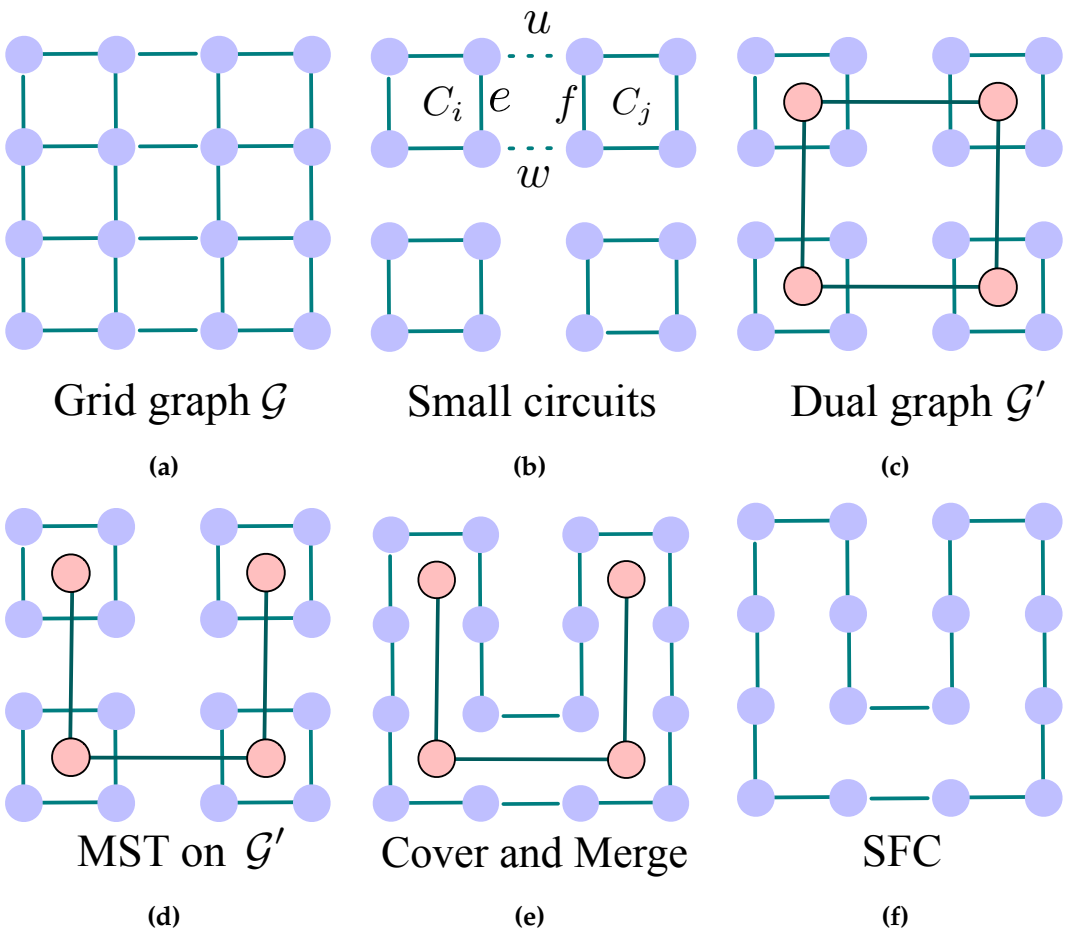


Figure 2.1: *The process of generating data-adaptive SFCs.*

Different from fractal SFCs that iteratively generate curves, in data-adaptive SFC, the curve generation becomes a **Cover-and-Merge** process [70, 17, 102].

During the **cover** process, a grid graph $\mathcal{G}$ is first constructed, which is an undirected graph and the nodes in $\mathcal{G}$ are the pixels as shown in Figure 2.1a. Then, many small circuits are constructed by separating the grid graph $\mathcal{G}$ as illustrated in Figure 2.1b. After that, we construct $\mathcal{G}'$, a dual graph of $\mathcal{G}$, the process can be found in Figure 2.1c. In the final step, we assign the edge weights of $\mathcal{G}'$ based on the distribution of data. We provide Dafner [17] SFC as an example to show how to generate a data-adaptive SFC. In this method, we first construct the grid graph

and dual graph as mentioned before, then we use the image context to determine $\mathcal{G}'$. Specifically, in $\mathcal{G}'$, suppose the edge weight between two nodes C_j and C_i is $\boldsymbol{W}(C_i, C_j)$, it will be calculated by using the pixel differences:

$$\boldsymbol{W}(C_i, C_j) = |u| + |w| - |e| - |f|,$$

Here f, w, u, e are pixel differences in four directions, for example, $|u| = |p_2 - p_1|$ as depicted in Figure 2.1b. Also, we can use deep neural networks to determine $\boldsymbol{W}(C_i, C_j)$ like NSFC that was proposed by Wang *et al*. [102]. In NSFC, for an input image, the edge weights $\boldsymbol{W}$ in $\mathcal{G}'$ are given by:

$$\boldsymbol{W} = \mathrm{E}(\mathrm{G}(\mathrm{Conv}(\boldsymbol{\mathcal{I}}))),$$

Where $\mathrm{F}(\cdot)$ is the CNN backbone to extract image features and $\mathrm{G}(\cdot)$ is a GNN that determines the $\boldsymbol{W}$. After obtaining $\boldsymbol{W}$, they provide an evaluator network $\mathrm{E}(\cdot)$ to identify whether it is a good SFC according to the given metric.

Until now, we have a dual graph $\mathcal{G}'$ with weights $\boldsymbol{W}$, then we can go to the **merge** step. In the first stage of merge, we will find a minimum-spanning tree τ according to the $\boldsymbol{W}$ in $\mathcal{G}'$ as shown in Figure 2.1d. Next, small circuits that are generated by $\mathcal{G}$ are connected if two adjacent small circuits are linked by τ as demonstrated in Figure 2.1e. Finally, we remove the MST and get a desired SFC in Figure 2.1f.

2.3 Efficent Point Cloud Segmentation

Image-based Point Cloud Analysis. Recently, many researchers are trying to introduce 2D neural networks designed for images to point cloud analysis, as several benefits can be obtained from 2D models. Firstly, 2D convolution is more efficient than 3D convolution when handling voxel data. Secondly, 2D models are

well-explored, which means there are a lot of methods that can be applied directly to enhance performance, such as attention, pre-trained models, etc. This benefit has already been proven by previous works [96, 111, 22, 25]. However, due to the heterogeneous nature of point cloud data and image data, 2D networks cannot be applied to point clouds directly. Therefore, the mapping from point cloud to 2D image is the most important step.

For example, the point-to-image mapping in MVCNN [96] involves collecting snapshot images from different views of a point cloud and using these images for classification. However, this method cannot capture the spatial features in the point cloud and thus cannot perform point cloud segmentation. Another point-to-image method is to use range images, such as in RangeNet++ [71] and SqueezeSeg [109]. The range image method makes point cloud segmentation in image space possible, but the spatial information of the original 3D data is inevitably weakened during 2D projection.

Currently, the most popular method for point-to-image mapping is to apply a bird's eye view image, as mentioned in PolarNet [129]. It converts LiDAR point clouds to 2D images using polar coordinates. This mapping approach achieves good performance in LiDAR point cloud tasks such as segmentation and detection. In our method, we propose a novel point-to-image mapping approach. Specifically, the mapping process is conducted by using a Hilbert curve to flatten the original 3D voxel into 2D. Due to the locality-preserving property of the Hilbert curve, the flattened voxel retains the same spatial information as its original 3D shape.

Point Cloud with Space Filling Curve. Space-filling curves [73] can fill all the 2D or 3D spaces within a single line. This property makes them good data linearization methods. Common space-filling curves like the Hilbert curve [33], Z-order curve [75],

and sweep curve are widely used in GIS or database applications. SFCs can also be used in point cloud analysis, and previous works usually use them together with tree structures. Prior work [103] partitions the original point cloud using OCTree and then applies the Hilbert curve to compress the data, while C-Flow [81] uses the Hilbert curve to reorder the point cloud for better feature clustering. O-CNN [104] combines the Z-curve with OCTree for faster data query and accelerates point cloud analysis. In our research, we use SFCs in a different way. We do not use SFCs to compress input data, and hence we do not use tree structures when applying SFCs. Instead, we linearize the 3D voxel data via a space-filling curve directly for convolution acceleration. During our research, we will use the Hilbert curve because the sequence that is flattened by the Hilbert curve preserves the locality of its original shape.

Multi-View Fusion for Point Cloud Analysis. Point clouds can be represented by 1D point data, 2D vertex data, and 3D voxel data. Based on these different formats, several approaches have been proposed [35, 15, 141, 133] for better point cloud recognition. However, each type of representation has its own limitations. By integrating these different formats together, we can not only benefit from the strengths of each, but also mitigate their weaknesses. For example, Cylinder3D [138, 142] and PVCNN [66] combine the 3D voxel feature with the 1D point feature. This operation lowers the computational cost produced by 3D voxel and makes 1D point prediction more accurate. In addition to these 1D-3D fusion models, 2D-3D models are also proposed for better point cloud analysis, such as MCVNN [83], Image2Point [117]. These approaches successfully utilize the pre-training model in 2D convolutional networks for 3D point cloud scenarios. Different from previous methods, our proposed method provides a novel 1D-2D fusion pipeline, which further reduces the computational overhead compared to the 1D-3D model. Also,

we can use knowledge suitable for 2D networks to improve our model, similar to 2D-3D fusion methods.

Chapter 3

Efficient Image Segmentation

3.1 Methodology

3.1.1 Overview

Context information is an important element if we want to predict semantic information from a featuremap. However, if modeling context information into a tensor, considering the complexity of context information, it must be a high-rank tensor. The modeling process will be very expensive.

Tensor decomposition is one of the solution to the context modeling, which can reconstruct high-rank tensor from a series of low-rank tensor. Since low-rank tensor is much cheaper to modeling, it would be an ideal solution to context modeling problem. Among the tensor decomposition methods, we are more interested in CP decomposition, which break the high-rank tensor into multiple rank-1 tensor. In our application, we do not predict context information directly. Instead, we predict multiple rank-1 fragments of original context feature and reconstruct them into the original tensor. This modeling scheme not only preserve the original channel-

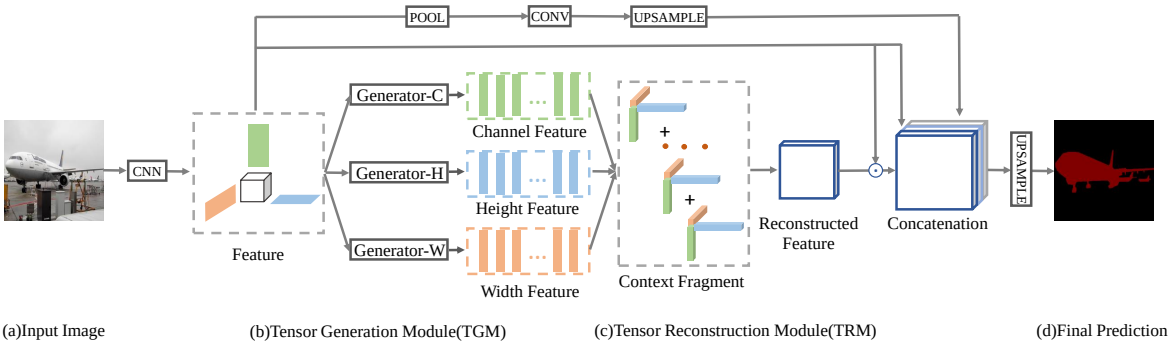


Figure 3.1: *The framework of RecoNet includes two key components: the Tensor Generation Module, which implements the generation of rank-1 tensors, and the Tensor Reconstruction Module, which implements tensor CP reconstruction.*

wise and spatial-wise feature, but also addresses the challenge posed by high-rank complexity.

The pipeline of our model is shown in Figure 3.1, which is composed by a low-rank tensor generation module (TGM) and a high-rank tensor reconstruction module (TRM). Additionally, we introduced a global pooling module (GPM) for global information collection. Similar to the FCN pipeline, we upsample the final output before prediction.

Formulation: A typical rank r CP reconstruction is formulated as below.

$$\boldsymbol{A} = \sum_{i=1}^{r} \lambda_i (\boldsymbol{vc}_i \otimes \boldsymbol{vh}_i \otimes \boldsymbol{vw}_i), \tag{3.1}$$

Where $\boldsymbol{vc}_i \in \mathbb{R}^{C\times 1\times 1}$, $\boldsymbol{vh}_i \in \mathbb{R}^{1\times H\times 1}$ and $\boldsymbol{vw}_i \in \mathbb{R}^{1\times 1\times W}$. λ_i is introduced to adjust the weight of each element.

3.1.2 Tensor Generation Module

In this section, we will introduce the key parts of TGM. To make the illustration clearer, we will first define what is context fragements, then we will illustrate how to use them to implement TGM.

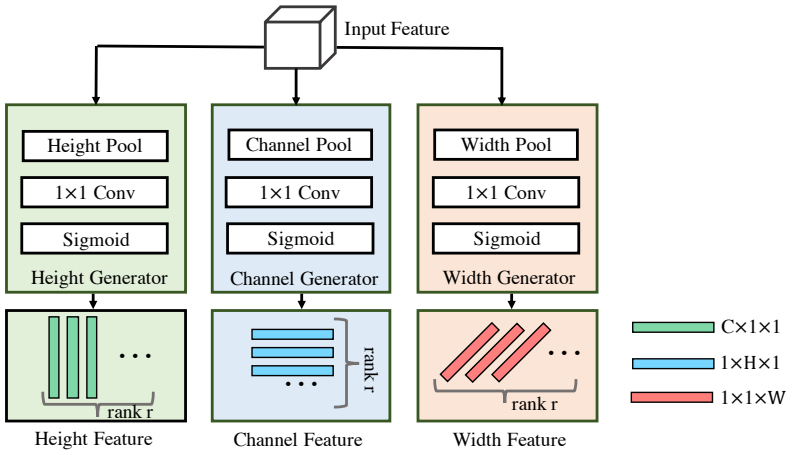


Figure 3.2: *Details of Tensor Generation Module. We perform average pooling at width, height, and channel dimension and marked as Width Pool, Height Pool and Channel Pool.*

Feature Generator. Featrue generator is a series of 1×1 convolution along height, width and channel direction. Each feature is calculated by a global pooling and a randomly initialized convolution. The weights of convolutions are not shared, which ensure that every generated feature learns different part of context information.

Context Fragments. We define the output tensor of the feature generator as the context fragment. Obviously, each fragment contains a small part of context information. Every context fragment is a 1-dimensional tensor in C/H/W direction and they are formulated as vc_i, vh_i, and $vw_i, i \in r$, as previously defined. The feature generator works r times and we will obtain $3r$ context fragments. The detailed information can be found in Figure 3.2.

Non-linearity in TGM. We introduce non-linearity into the Tensor Generation Module (TGM) by applying the Sigmoid function to the context fragments, a process akin to the unary attention scheme. This operation serves a dual purpose. Firstly, it regularizes the values in the context fragments to fall within the range [0, 1], thereby enhancing the stability of the output. Secondly, it boosts the representation capacity of our network, allowing it to capture more complex patterns and relationships in

the data. This addition of non-linearity is a crucial aspect of our model, contributing significantly to its performance and robustness.

3.1.3 Tensor Reconstruction Module

In this section, we will introduce how to reconstruct the context fragments into a high-rank tensor. The working flow of tensor reconstruction module can be found in Equation (3.1). In the following context, we will introduce TRM in detail and introduce why using this reconstruction shceme.

Context Aggregation. After getting 3 context fragments, we reconstruct them into an attention map $\boldsymbol{A} = \{a_1, a_2, \ldots, a_{CHW}\}$ using Equation (3.1). Suppose we have a featruemap $\boldsymbol{X} = \{x_1, x_2, \ldots, x_{CHW}\}$, the context featuremap $\boldsymbol{Y} = \{y_1, y_2, \ldots, y_{CHW}\}$ is formulated as:

$$\boldsymbol{Y} = \boldsymbol{A} \cdot \boldsymbol{X} \iff y_i = a_i \cdot x_i, i \in CHW. \tag{3.2}$$

This formulation reveals a new way to generate attention map, different from previous spatial attention [132] or channel attention [125] mechanism, our new attention mechanism can obtain both spatial and channel context feature.

Low-rank Reconstruction. The tensor low-rank-to-high-rank reconstruction process can be found in Figure 3.3. Firstly, we use context fragements $\boldsymbol{v}c_i \in \mathbb{R}^{C\times1\times1}$, $\boldsymbol{v}h_i \in \mathbb{R}^{1\times H\times1}$ and $\boldsymbol{v}w_i \in \mathbb{R}^{1\times1\times W}$ to cinstruct a sub-attention map $\boldsymbol{A}_i$ via Equation (3.2). Then we sum up all the sub-attention map using Equation (3.3) to a high rank attention map $\boldsymbol{A}$.

$$\boldsymbol{A} = \sum_{i=1}^{r} \lambda_i \boldsymbol{A}_i. \tag{3.3}$$

Where a learnable factor$\lambda_i \in (0, 1)$ is introduced to normalize the attetion map.

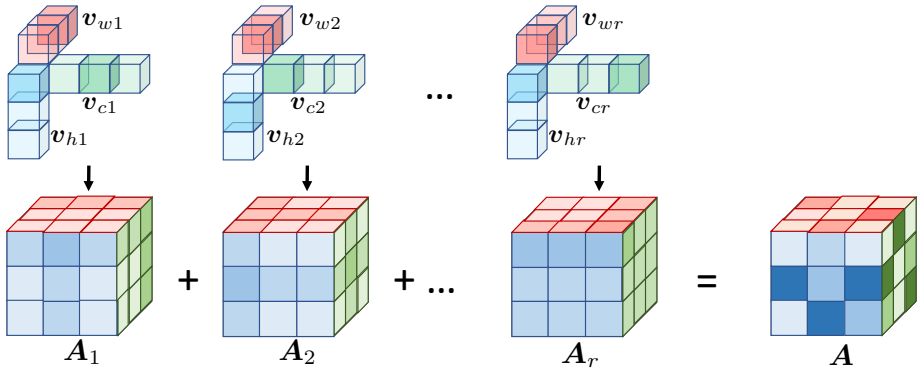


Figure 3.3: *The details of Tensor Reconstruction Module. In this module, we first generate a series of sub-attention map* $A1, A2, \ldots, Ar$ *using width / channel / height context fragments that come from TGM and the process is marked with* $\downarrow$*. After that, we sum them up to implement context CP reconstruction.*

3.1.4 Global Pooling Module

Similar to previous works [126, 131, 8], we also adopt global average pooling to boost the network performance. It is a combination of Pool-Conv1 $\times$ 1 and we name this module as Global Pooling Module (GPM).

3.1.5 Network Details

We apply dilated ResNet as our backbone, which is the same as [125] and the output feature of backbone will then go through TGM and TRM for context information collection. Auxiliary loss that applied to the Res-4 output is proved to be useful in prior works [125, 131] and hence we add this design into our framework. The objective of our model is then show as:

$$\mathcal{L} = \mathcal{L}main + \alpha\mathcal{L}aux. \tag{3.4}$$

Here we adjust the auxiliary loss weight α to 0.2.

3.2 Experiments

In this part, we reported the experimental results of RecoNet on five popular dataset: SIFT-FLOW, ADE20K, COCO-Stuff, PASCAL-Context and PASCAL-VOC12.

3.2.1 Implementation Details

The batch size for all experiments is set to 16 in the RecoNet implementation, and we randomly flip and rescale the input image from 0.5 to 2, then crop a 512×512 patch from the scaled image as the input, which is the same as in [131, 125]. This setting requires a multi-GPU infrastructure since a single GPU does not have enough memory to load all images from a batch. To solve this problem, we use syncBN [26, 125], which was applied in previous designs. It allows batch normalization to be implemented across GPUs. During the PASCAL-VOC12, PASCAL-Context, and COCO-Stuff experiments, we initialized the learning rate with the value 0.001, while the initial learning rates for the ADE20K and SIFT-FLOW datasets are 0.01 and 0.0025, respectively. The learning rate will decay following the equation $lr = base\_lr \times (1 - \frac{iter}{total\_iters})^{power}$, where the $power$ is 0.9. The training process will last 120 and 180 epochs for the ADE20K and COCO-Stuff datasets, while the number of training epochs we set for other datasets is 80. RecoNet uses the Pytorch [77] framework, and the optimizer we use is SGD with a weight decay of 1e-4 and momentum of 0.9.

In the inference stage, we apply a multi-scale strategy, which resizes the input image to 0.75, 1, 1.25, 1.5, 1.75, 2.0] of the original scale, and the output feature of each scale is summed up for the final prediction. The main metric for all experiments is the mean intersection-over-union (mIoU).

Table 3.1: *Performance of different segmentation models on PASCAL-VOC12 dataset without COCO-pretraining.*

	PSPNet[131]	FCN[68]	APCNet[31]	DMNet[30]	CFNet[126]	EncNet[125]	RecoNet
cat	95.9	77.6	96.0	96.4	95.9	94.2	**96.5**
bike	71.9	34.2	75.8	**77.3**	71.9	69.2	66.3
plant	72.8	45.2	75.8	76.6	74.9	68.7	**78.1**
bus	95.2	75.3	96.9	**97.1**	94.8	96.3	94.5
bottle	75.8	60.3	80.6	78.1	82.8	86.2	**87.4**
horse	94.5	63.9	95.0	95.8	94.6	92.5	**95.9**
car	89.9	74.7	90.0	**92.7**	90.0	90.7	92.6
aero	91.8	76.8	95.8	**96.1**	95.7	94.1	93.7
chair	39.3	21.4	42.0	39.8	37.1	38.8	**48.4**
cow	90.7	62.5	93.7	91.4	92.6	90.7	**94.5**
dog	90.5	71.8	91.6	92.7	93.4	90.0	**94.4**
table	71.7	46.8	75.4	75.5	73.0	73.3	**76.6**
bird	94.7	68.9	84.5	94.1	95.0	**96.3**	95.6
boat	71.2	49.4	76.0	72.8	76.3	**76.7**	72.8
mbike	88.8	76.5	90.5	91.0	89.6	88.8	**93.8**
person	89.6	73.9	89.3	90.3	88.4	87.9	**90.4**
sheep	89.6	72.4	92.8	94.1	**95.2**	92.6	93.6
sofa	**64**	37.4	61.9	62.1	63.2	59.0	63.4
train	85.1	70.9	88.9	85.5	**89.7**	86.4	88.6
tv	76.3	55.1	79.6	77.6	78.2	73.4	**83.1**
mIoU	82.6	62.2	84.2	84.4	84.2	82.9	**85.6**

3.2.2 Results on Different Datasets

PASCAL-VOC12. The PASCAL-VOC12 dataset [23] has 21 categories, 10,582 images for training (including the PASCAL augmentation dataset), 1,449 images for validation, and 1,456 images for testing. The comparison between RecoNet and other methods can be found in Table Table 3.1, which shows that our proposed RecoNet achieves an mIoU of 85.6%, outperforming the latest state-of-the-art DMNet [30] by a large margin.

Next, we perform COCO-pretraining for the PASCAL-VOC12 challenge. Specif-

Table 3.2: *Performance of different segmentation models on PASCAL-VOC12 dataset with COCO-pretraining.*

Method	Backbone	mIoU
CRF-RNN [135]		74.7
DPN [67]		77.5
Piecewise [60]		78.0
ResNet38 [114]		84.9
PSPNet [131]	ResNet-101	85.4
DeepLabv3 [8]	ResNet-101	85.7
EncNet [125]	ResNet-101	85.9
DFN [123]	ResNet-101	86.2
CFNet [126]	ResNet-101	87.2
EMANet [54]	ResNet-101	87.7
DeeplabV3+ [9]	Xception	87.8
DeeplabV3+ [9]	Xception+JFT	89.0
RecoNet	ResNet-101	**88.5**
RecoNet	ResNet-152	**89.0**

Table 3.3: *Performance of different segmentation models on PASCAL-Context dataset (background is included).*

Method	Backbone	mIoU
FCN-8s [68]		37.8
ParseNet [65]		40.4
Piecewise [60]		43.3
VeryDeep [113]		44.5
DeepLab-v2 **[SEGM-TPAMI2018-Deeplab]**	ResNet-101	45.7
RefineNet [59]	ResNet-152	47.3
PSPNet [131]	ResNet-101	47.8
MSCI [58]	ResNet-152	50.3
Ding *et al.* [21]	ResNet-101	51.6
EncNet [125]	ResNet-101	51.7
DANet [26]	ResNet-101	52.6
SVCNet [20]	ResNet-101	53.2
CFNet [126]	ResNet-101	54.0
DMNet [30]	ResNet-101	54.4
RecoNet	ResNet-101	**54.8**

Table 3.4: *Performance of different segmentation models on COCO-Stuff dataset.*

Method	Backbone	mIoU
FCN-8s [68]		22.7
DeepLab-v2 **[SEGM-TPAMI2018-Deeplab]**	ResNet-101	26.9
RefineNet [59]	ResNet-101	33.6
Ding *et al.* [21]	ResNet-101	35.7
SVCNet [20]	ResNet-101	39.6
DANet [26]	ResNet-101	39.7
EMANet [54]	ResNet-101	39.9
RecoNet	ResNet-101	**41.5**

Table 3.5: *Performance of different segmentation models on SIFT-Flow dataset.*

Method	pixel acc.	mIoU
Sharma et al. [93]	79.6	-
Yang et al. [120]	79.8	-
FCN-8s [68]	85.9	41.2
DAG-RNN+CRF [94]	87.8	44.8
Piecewise [60]	88.1	44.9
SVCNet [20]	89.1	46.3
RecoNet	**89.6**	**46.8**

Table 3.6: *Performance of different segmentation models on ADE20K validation dataset.*

Method	Backbone	mIoU
RefineNet [59]	ResNet-152	40.70
PSPNet [131]	ResNet-101	43.29
DSSPN [57]	ResNet-101	43.68
SAC [90]	ResNet-101	44.30
EncNet [125]	ResNet-101	44.65
CFNet [126]	ResNet-50	42.87
CFNet [126]	ResNet-101	44.89
CCNet [38]	ResNet-101	45.22
RecoNet	ResNet-50	**43.40**
RecoNet	ResNet-101	**45.54**

ically, we follow the training process mentioned in previous works [125, 31, 126, 30, 26], pretraining RecoNet using the COCO [62] dataset with a learning rate of 0.004 and a total of 30 training epochs. Then, we follow the implementation details as mentioned before, training the PASCAL-VOC12 dataset including the PASCAL augmentation data with a learning rate of 0.001 and 80 epochs. Finally, we fine-tune all the training and validation data with an additional 50 epochs and use the model for testing. The results are evaluated by the PASCAL-VOC12 server and the performance is shown in Table 3.2. Our model gets 89.0% mIoU, which is

the highest score on the leaderboard.[1]. The result demonstrates that our low-rank reconstruction strategy is powerful enough to capture context information.

PASCAL-Context. The PASCAL-Context dataset [120] is more challenging than the PASCAL-VOC12 dataset as it contains 4,998 training images and 5,105 testing images, with a total of 60 categories including foreground objects and background stuff. Table 3.3 reports the performance of RecoNet on this dataset. For a fair comparison, we set the evaluation method to be identical to previous studies [126, 125, 31], which take unlabeled data into account. Our RecoNet achieves an mIoU of 54.8%, which significantly surpasses previous works given the difficulty of the dataset. Notably, our method has the best performance among the attention-based methods like CFNet, DMNet, and DANet.

COCO-Stuff. The COCO-Stuff dataset [6] is derived from the MS-COCO dataset, which has a total of 10K images (9K for training, 1K for testing). It is labeled with 171 things and stuff, and the performance of different methods on this dataset can be found in Table 3.4. RecoNet achieves an mIoU of 41.5%, which outperforms all previous works that utilize context information for prediction.

SIFT-Flow. The SIFT-Flow dataset [64] is a relatively small and older dataset that mainly focuses on urban scenarios. It only contains 2,488 training images and 500 testing images, all of which are 256×256 in size. The dataset is heavily labeled with 33 classes. We tested RecoNet on this dataset to verify its scalability. The results in Table 3.5 demonstrate that RecoNet is suitable for a diverse range of datasets.

[1] http://host.robots.ox.ac.uk:8080/anonymous/PXWAVA.html

ADE20K. The ADE20K dataset [137] is an increasingly popular scene parsing benchmark since it contains more than 25K annotated images, making it one of the largest semantic segmentation datasets. All the images are categorized into 150 classes, including indoor, outdoor, and urban scenes, etc. The results of different models on this dataset can be found in Table 3.6. Our RecoNet consistently performs better than other attention-based methods like CCNet, CFNet, and EncNet, regardless of the backbone used. This further substantiates our claim that RecoNet can model context information better than previous attention methods.

3.2.3 Ablation Study

In this section, we aim to assess the contribution of each part of RecoNet to the final performance. Additionally, we will study the influence of the tensor reconstruction rank r. The PASCAL-VOC12 training and validation datasets are used during the ablation study, and all reported results are on the VOC12 validation dataset.

Component Evaluation. Here, we create multiple segmentation models with different components of RecoNet, including TGM, TRM, the global pooling module (GPM), and auxiliary loss, to measure the importance of each component to the final prediction accuracy. The other settings, including tensor rank ($r = 64$) and implementation details, are identical to those mentioned in previous sections. The ablation study of each part can be found in Table 3.7. It shows that TGM+TRM contributes a 9.9% mIoU with the ResNet-50 backbone, which is the most important part, while the GPM contributes another 0.6% mIoU improvement. This result demonstrates that the TGM+TRM design is the most significant component and that the GPM is not trivial. It also implies that the context modeling method proposed by our RecoNet is powerful. Additionally, the auxiliary loss further boosts the model by 0.6% with the ResNet-50 backbone. The final result we obtained has an mIoU

Table 3.7: *Ablation experiments of RecoNet. The model is tested on the PASCAL-VOC12 val set. FT is the short of PASCAL-VOC12 fine-tuning.*

Method	FT	Aux-loss	GPM	TGM+TRM	MS/Flip	mIoU %
ResNet-50						68.7
ResNet-50				√		78.6
ResNet-50			√	√		79.2
ResNet-50	√		√	√		79.8
ResNet-101		√		√		81.4
ResNet-101	√		√	√	√	82.1
ResNet-101	√	√	√	√	√	82.9

of 82.9%, which is the combination of each block coupled with some tricks like multi-scale, flip test, and fine-tuning.

Tensor Rank. The tensor rank r determines the capacity of information that we can reconstruct from context fragments, which is essential to the final performance. Intuitively, a larger r leads to a larger capacity and results in better performance. However, a larger r also increases the risk of overfitting and may harm prediction accuracy. Therefore, we manually set the rank r from 16 to 128 to find the best parameter. The results of different r values can be found in Table 3.8, which indicates that $r = 64$ is the optimal parameter. The reason is that the input tensor of TGM X has the shape of $512 \times 64 \times 64$, which implies the tensor rank is 64. Hence, when r is smaller than 64, the mIoU increases simultaneously, but the accuracy goes down when r is larger than 64.

Contrasting with Earlier Methods. To show the effectiveness of RecoNet, we re-implemented multiple state-of-the-art researches and compare them with our model in terms of GFLOPs and mIoU. The baseline ResNet-101 is the backbone we used during our research, which substitute the first convolution module with kernel size 7 to 3 convolutions with kernel 3 like previous works [131, 125, 126, 38, 54]. Also, the dilated convolution is applied to the backbone for larger receptive field. Since the performance of our backbone already outperforms some models according to their

Table 3.8: *Ablation experiments of tensor rank. We use ResNet101 as the backbone during experiments and the results are obtained after multi-scale test.*

Method	Tensor Rank	mIoU %
RecoNet	16	81.2
RecoNet	32	81.8
RecoNet	48	81.4
RecoNet	**64**	**82.1**
RecoNet	80	81.6
RecoNet	96	81.0
RecoNet	128	80.7

Table 3.9: *Comparison of different segmentation models on PASCAL-VOC12 validation dataset. Our proposed RecoNet has the best mIoU score with relatively low computational cost.*

Method	SS	MS/Flip	FLOPs
ResNet-101	-	-	190.6G
DeepLabV3+ [9]	79.45	80.59	+84.1G
PSPNet [131]	79.20	80.36	+77.5G
DANet [26]	79.64	80.78	+117.3G
PSANet [132]	78.71	79.92	+56.3G
CCNet [38]	79.51	80.77	+65.3G
EMANet [54]	80.09	81.38	+43.1G
RecoNet	**81.40**	**82.13**	**+41.9G**

original paper. To make a fair comparison, we use our backbone and implementation settings for all methods. The comparison can be found in Table 3.9 and we can find that since RecoNet harvests global context information, the performance is better than PSPNet [131] and DeepLabV3+ [9], which use large kernel to expand receptive field. Additionally, we RecoNet has lower computational cost than DANet [26] and PSANet [132] since the attention in RecoNet is constructed by low rank tensor while DANet [26] and PSANet [132] use vanilla attention, which is costly. Also, our method shows better mIoU compared with the efficient attention methods like CCNet [38] and EMANet [54]. This is because RecoNet collects both spatial and channel attention simultaniously while other methods can only collect context information in spatial or channel only.

3.2.4 Further Discussion

In this section, we measure the computational cost of RecoNet to illustrate the effectiveness of our design. Next, we visualized some sub-attention maps to demonstrate that each sub-attention map will learn a part of context feature.

Analysis of Computational Complexity. The attention map of our model is generated by the context low-rank reconstruction, which has smaller computational

Table 3.10: *The floating point operations and GPU memory occupations of different attention module. Here we set $r = 64$ for this experiment.*

Method	Channel	GPU Memory	FLOPs
A^2Net [12]	512	25.00MB	4.30G
Non-Local [106]	512	88.00MB	19.33G
EMAUnit [54]	512	24.12MB	2.42G
LatentGNN [128]	512	44.69MB	2.58G
APCNet [31]	512	193.10MB	8.98G
RCCA [38]	512	41.33MB	5.37G
AFNB [143]	512	25.93MB	2.62G
TGM+TRM	512	**8.31MB**	**0.0215G**

cost compared with other approaches. The FLOPs of our model is $\mathcal{O}(C^2 + H^2 + W^2 + CHW) \approx \mathcal{O}(CHW)$, where the $\mathcal{O}(CHW)$ comes from tensor low-rank-to-high-rank reconstruction and $\mathcal{O}(C^2 + H^2 + W^2)$ is caused by context fragment generation. Recalling that the FLOPs of generating and vanilla attention map is $\mathcal{O}(CH^2W^2)$. Hence, the cost of TGM+TRM is significantly small. For example, on the table Table 5.3, we can find that our design is **900** times faster than non-local attention. Also, our method is over **100** times faster than other efficient attention mechanism like EMAUnit [54].

Visualization. To find out the performance and the function of each sub-attention map, we conduct a visualization experiment. Here we visualized some attention-activated featuremap X, which has the formulation of $\boldsymbol{A}_i \cdot \boldsymbol{X}$. The results can be found in Figure 3.4.

Limitation. Our method, although significantly improved in terms of accuracy and inference speed, still has drawbacks. It requires a large amount of storage space, typically 3-4 times that of other methods. This is because during the TGM process, we produce r rank-1 tensors, and each rank-1 tensor requires the generation of a fully connected layer to learn its features. Therefore, our method results in substantial memory consumption.

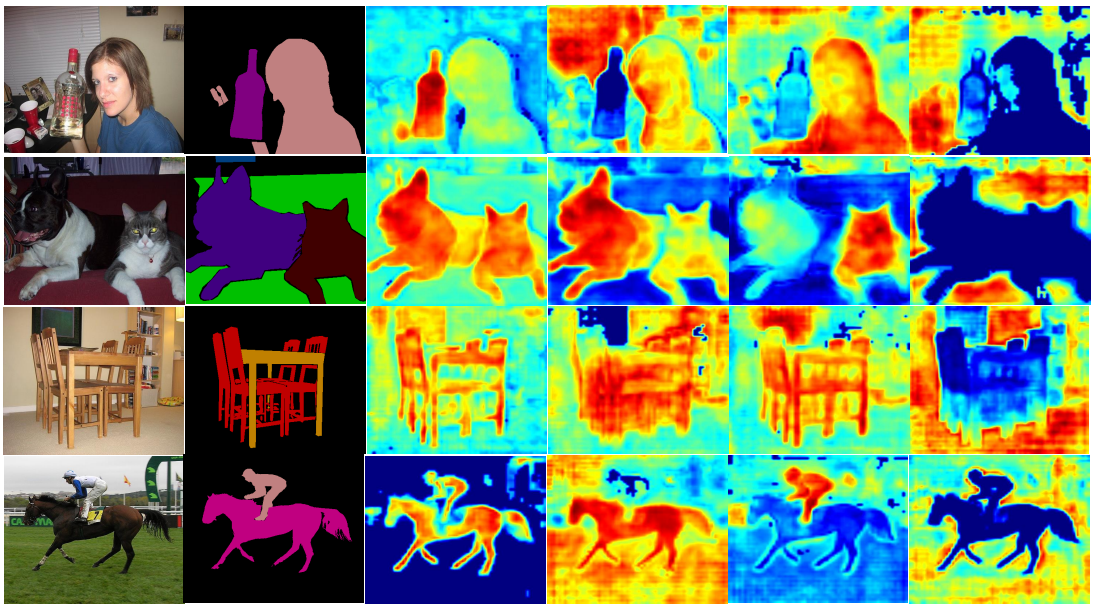


Figure 3.4: *Visualization of $\boldsymbol{A}_i \cdot \boldsymbol{X}, i \in 64$. Here $\boldsymbol{A}_i$ represents the sub-attention map. We provide the original images and their labels for reference. We can observe that each sub-attention map extracts a part of context information.*

3.3 Conclusion

In this part, we introduced how to implement image segmentation with high quality and low cost. For high quality segmentation, attention mechanism is strongly needed, but it will bring more computational cost. If we want to perform efficient segmentation, we should apply attention simplification technique. In this topic, our research proposed to use tensor low-rank reconstruction for attention acceleration. This approach not only reduces the computational cost of attention by a large margin, but also preserve the 3D context information of original featuremap, which brings high quality segmentation.

Future Work. We will subsequently explore efficient segmentation with Large Language Models (LLMs), especially interactive segmentation. Interactive segmentation

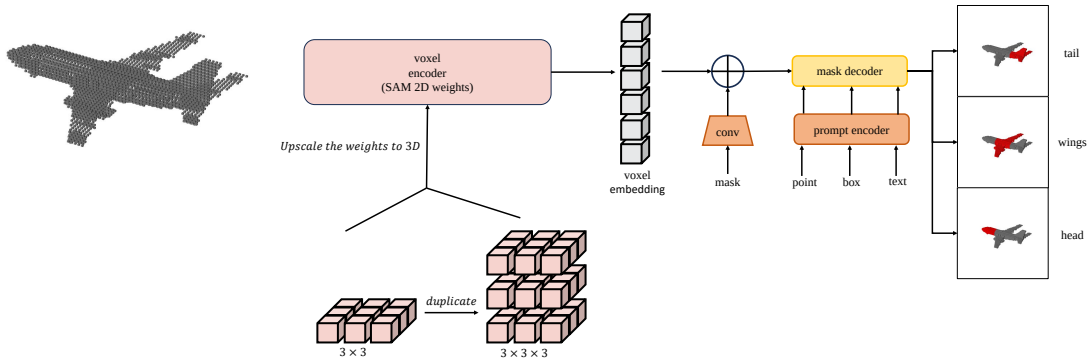


Figure 3.5: *The framework of our SAM 3D model. In this model, we will use the pre-trained 2D SAM model and upscale the weights to adapt 3D voxels. After the voxel encoder, the obtained voxel embedding will merge with several visual or text prompts to achieve segmentation in different parts. For example, we can use text prompt like "head", "wings" or "tail" to find different parts on the airplane voxel.*

refers to the process where humans interact with machines to achieve high-precision segmentation. Initially, a person provides a visual prompt, which is usually a mouse click or a text description. Then, the segmentation model outputs a corresponding segmentation mask based on the given prompt. Afterward, the person further modifies the prompt based on whether the generated mask meets the requirements, ultimately achieving high-precision segmentation. Large Language Models are currently a hot topic among researchers and their emergence has taken interactive segmentation to a new level. Previously, mouse clicks were often the only effective prompts but now with the help of LLMs, text prompts have also become effective. A representative interactive segmentation framework is SAM [44], which integrate LLMs to realize interactive 2D image segmentation.

In the future work, we will dedicate to produce 3D segment anything model, which can be used for meidial imaging and point cloud analysis. The working pipeline of our proposed model is shwon as follow: Here we simply use the open-source pre-trained weight from SAM and upscaling the weight to 3D scenario. For example, a convolution kernel with size 3×3 will be interpolated to $3 \times 3 \times 3$ and

then we fine-tuning the weight to adapt 3D data. Therefore, the voxel encoder and mask decoder are transformer encoder and decoder structure that comes from SAM model directly while the CLIP [87] model is used as our prompt encoder. However, the fine-tuning of 3D SAM model would be a challenging problem. Due to computational resource limitations, training a 3D SAM model directly is a very time-consuming approach. To address this issue, we use large model low-rank adaptation (LoRA) [36], which is formulated as:

$$W = W^0 + A \times B \tag{3.5}$$

Where the W^0 is the original weight parameters in LLMs. During the LoRA fine-tuning process, the original model is fixed and we only update the newly introduced parameters in matrix $A \in \mathcal{R}^{H \times r}$ and $B \in \mathcal{R}^{r \times W}$ and the rank r is manually adjusted. We will use LoRA technique to finetune the 2D SAM model and making it capable to represent 3D scenarios.

The work mentioned above is what I plan to continue researching after completing this thesis. Some of these studies will continue along my current research path such as space-filling curve applications, while others will explore new fields such as efficient LLM with segmentation. However, overall, they are all very challenging tasks. Currently, there is great potential for research in the acceleration of segmentation. In the future, I will focus most of my attention on research related to the acceleration of segmentation in 3D scenes.

Chapter 4

Space Filling Curve for Efficient Data Clustering

4.1 Methodology

In the previous section, we introduced how to accelerate segmentation from the perspective of attention slimming. However, the speedup that we can achieve is limited since the backbone network consumes most of the computational overhead. Therefore, we aim to find a method that is applicable for backbone acceleration. Here, we propose a novel technique, which involves using space-filling curves (SFCs) for acceleration. Since SFCs can be used to flatten the feature, we can then substitute the original Conv2D with Conv1D in the backbone network, which will greatly reduce the cost. To begin with, we will introduce how to apply a space-filling curve to a deep neural network.

4.1.1 Problem Definition

Generating a space-filling curve is not difficult, and it simply follows the process below. We first transform an image to grayscale with dimension $\boldsymbol{\mathcal{I}}^{\in 1\times H\times W}$. Then we separate the pixels in $\boldsymbol{\mathcal{I}}$ into multiple 2×2 circuits as depicted in 2.1 (b). Then a dual graph [5] $\mathcal{G}'$ is obtained as shown in 2.1 (c), where each edge has its own weight. Based on the value of each edge, a minimum spanning tree can be generated as shown in 2.1 (d). Finally, we use an algorithm called the Cover-and-Merge algorithm [70] to link all small circuits, and the SFC is obtained as shown in 2.1 (f).

Overview. Our SFC generation framework is shown in Figure 5.3, which is consists of two part, graph weight $\boldsymbol{W}\mathcal{G}'$ generation and SFC generation. In the first part, the process can be described as:

$$\boldsymbol{W}_{\mathcal{G}'} = \mathrm{EGCN}(\mathrm{L}(\mathrm{Conv}(\boldsymbol{\mathcal{I}}))). \tag{4.1}$$

Where a convolutional neural network is used as backbone to extract features from image and denoted as $\mathrm{Conv}(\cdot)$. Then the features are followed by feature normalization layer $\mathrm{L}(\cdot)$ and Efficient-GCN $\mathrm{EGCN}(\cdot)$ layer for graph weight generation. For the second stage, the entire progress is shown as follow:

$$\mathrm{SFC} = \mathrm{Cover\_and\_Merge}(\mathcal{T}(\boldsymbol{W}\mathcal{G}')). \tag{4.2}$$

where the Prim MST [79] is firstly constructed on $\boldsymbol{W}\mathcal{G}'$ and denoted as $\mathcal{T}(\cdot)$. Next, we apply the Cover-and-Merge algorithm to generate the output SFC. Notably, the framework cannot be optimized by gradient back propagation directly since the MST and Cover-and-Merge algorithm are not differentiable. Therefore, we propose a Siamese Network-based optimization algorithm to update the network parameters, which will be introduced in the following text.

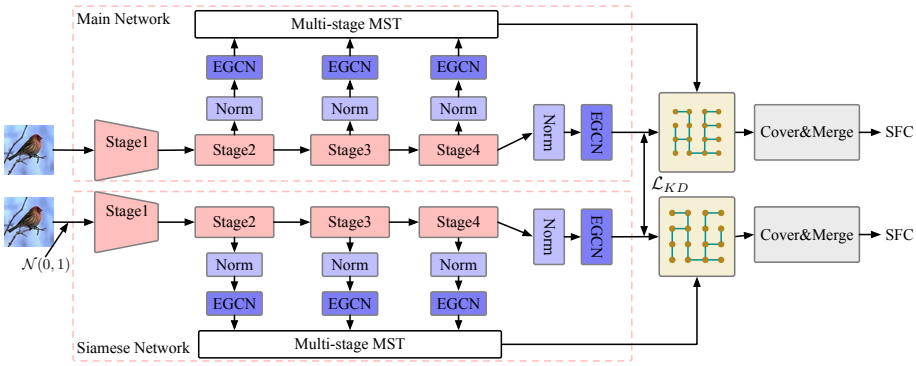


Figure 4.1: *The workflow of our proposed model. The Cover-and-Merge algorithm is marked as Cover&Merge, and* $\mathrm{L}(\cdot)$ *denotes our proposed feature normalization layer. We add Gaussian noise* $\mathcal{N}(0,1)$ *to the Siamese network input to increase input diversity.*

4.1.2 Graph Weight Generation

Convolutional Network. Our model accepts grayscale image input with size $\boldsymbol{\mathcal{I}}^{\in 1\times H\times W}$, and the image will be encoded by a residual network [32]. After that, we will interpolate the output feature of ResNet to its original size. Finally, we obtain the desired feature map $\boldsymbol{WI}^{C\times\frac{H}{2}\times\frac{W}{2}}$ by using a 2×2 convolution with stride 2. The entire process is simply described as:

$$\boldsymbol{\mathcal{X}} = \mathrm{F}(\boldsymbol{\mathcal{I}}). \tag{4.3}$$

Feature Normalization. After getting $\boldsymbol{W}_I$, we apply feature normalization, a useful technique that firstly proposed by GAT [101] to it for $\boldsymbol{W}\mathcal{G}_I$ generation. Specifically, we first reshape $\boldsymbol{W}_I$ into $\tilde{\boldsymbol{W}}_I \in \mathbb{R}^{N^2\times C}$, where $N = \frac{H}{2} = \frac{W}{2}$. Then we apply softmax across dimension C and dimension N^2 respectively, obtaining featuremap $(\mathrm{S}_1(\cdot))$ and $(\mathrm{S}_2(\cdot))$. The concatenation of them along C axies brings intermediate graph weights $\boldsymbol{W}\mathcal{G}_I$ to us as the description of the following formulation:

$$\boldsymbol{W}\mathcal{G}_I = \mathrm{L}(\tilde{\boldsymbol{W}}_I) = \mathrm{Concat}(\mathrm{S}_1(\tilde{\boldsymbol{W}}_I), \mathrm{S2}(\tilde{\boldsymbol{W}}_I)). \tag{4.4}$$

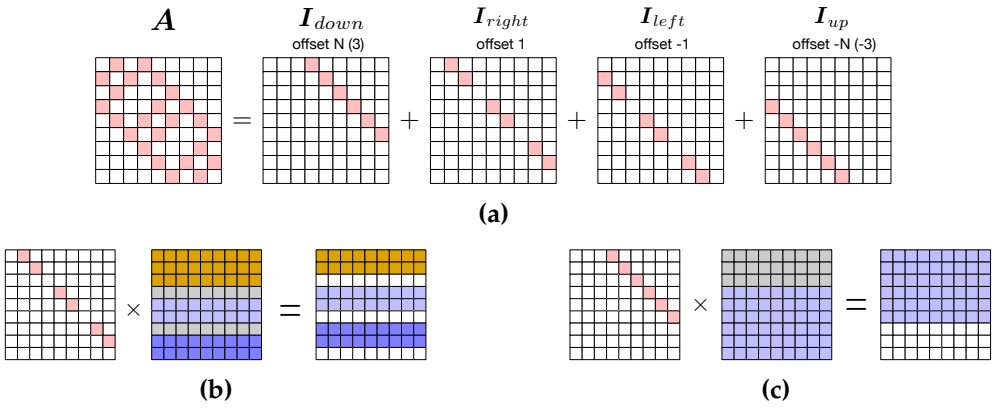


Figure 4.2: *Demonstration of offset-diagonal matrix multiplication. (a) The graph of a 3×3 image, which is a grid graph that consists of 4 offset diagonal matrices. (b) Multiplication with $\boldsymbol{I}_{right}$ is identical to moving all elements upward and replacing the grey rows to 0. (c) Multiplication with $\boldsymbol{I}_{down}$ is identical to moving all elements upward and padding the bottom rows. The white-colored elements represent 0, while others like gray, purple, blue, pink, and yellow indicate non-zero elements.*

$\boldsymbol{W}\mathcal{G}_I$ will be used as the input of EGCN, which is an efficient graph neural network algrithm that will be introduced in the following paragraph.

Efficient-GCN. EGCN is an GCN acceleration algorithm that specially designed for grid graph and it has the identical formulation to GCN like Equation (4.5). In our research, we will use the EGCN to gather global context information of a featuremap and predict edge weight $\boldsymbol{W}\mathcal{G}'$.

$$\begin{aligned} \boldsymbol{W}{\mathcal{G}_I}^{r+1} &= \sigma\left(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{W}{\mathcal{G}_I}^{r}\boldsymbol{W}k^{r}\right), \\ \boldsymbol{W}\mathcal{G}' &= \boldsymbol{W}{\mathcal{G}_I}^{R+1}, \quad r = 1, 2, \ldots, R, \end{aligned} \tag{4.5}$$

For the *rth* layer, we input the feature $\boldsymbol{W}{\mathcal{G}_I}^{r}$ and multiply it with an adjacency matrix $\hat{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A}$, $\boldsymbol{A}$, and a degree matrix $\hat{\boldsymbol{D}}$. The feature map is learned by the parameter update on $\boldsymbol{W}k^{r}$. Finally, the activation function $\sigma(\cdot)$ is used at the end of the *rth* layer. In traditional GCN, this process is actually time-consuming since the adjacency matrix has the size $\boldsymbol{A} \in \mathbb{R}^{HW \times HW}$. However, in our EGCN, the scale of the adjacency matrix is not a problem anymore because the EGCN accelerates the

vanilla GCN based on the distribution of $\boldsymbol{A}$ in the grid graph.

In grid graph, all nodes are connected by its left,right up and down node and then we can separate the adjacency matrix $\boldsymbol{A}$ of a grid graph into four parts as:

$$\begin{aligned}\boldsymbol{A} &= \boldsymbol{I}_{up} + \boldsymbol{I}_{down} + \boldsymbol{I}_{left} + \boldsymbol{I}_{right} \\ &= \boldsymbol{I}_{down} + \boldsymbol{I}_{right} + \boldsymbol{I}^{\top}{}_{down} + \boldsymbol{I}^{\top}{}_{right},\end{aligned}$$

The vector $\boldsymbol{I}_{left} \in \mathbb{R}^{N^2 \times N^2}$ and $\boldsymbol{I}_{up} \in \mathbb{R}^{N^2 \times N^2}$ represent the leftward and upward edges that connect each node respectively. The transpose of $\boldsymbol{I}_{left} \in \mathbb{R}^{N^2 \times N^2}$ and $\boldsymbol{I}_{up} \in \mathbb{R}^{N^2 \times N^2}$ are marked as $\boldsymbol{I}_{right} \in \mathbb{R}^{N^2 \times N^2}$ and $\boldsymbol{I}_{down} \in \mathbb{R}^{N^2 \times N^2}$. The relationship between each part can be found in Figure 4.2a. According to this separation, we can simplify the process $\boldsymbol{A}\boldsymbol{W}^r_{\mathcal{G}_I}$ from matrix multiplication to matrix addition as follow:

$$\begin{aligned}\boldsymbol{A}\boldsymbol{W}^r_{\mathcal{G}_I} &= \boldsymbol{I}_{down}\boldsymbol{W}^r_{\mathcal{G}_I} + \boldsymbol{I}_{right}\boldsymbol{W}^r_{\mathcal{G}_I} \\ &\quad + \boldsymbol{I}^{\top}{}_{down}\boldsymbol{W}^r_{\mathcal{G}_I} + \boldsymbol{I}^{\top}{}_{right}\boldsymbol{W}^r_{\mathcal{G}_I} \\ &= \boldsymbol{W}^r_{\mathcal{G}_{IN}} + \boldsymbol{W}^r_{\mathcal{G}_{I1}} + \boldsymbol{W}^r_{\mathcal{G}_{I-N}} + \boldsymbol{W}^r_{\mathcal{G}_{I-1}},\end{aligned} \tag{4.6}$$

The detailed process of equation $\boldsymbol{I}_{down}\boldsymbol{W}^r_{\mathcal{G}_I} = \boldsymbol{W}^r_{\mathcal{G}_{IN}}$ can be found in Figure 4.2c, which is simply shifting and padding the original matrix $\boldsymbol{W}^r_{\mathcal{G}_I}$ upward by N rows. Similar to $\boldsymbol{W}^r_{\mathcal{G}_{IN}}$, $\boldsymbol{W}^r_{\mathcal{G}_{I-N}}$ represents shifting and padding the original matrix $\boldsymbol{W}^r_{\mathcal{G}_I}$ downward by N rows. For $\boldsymbol{W}^r_{\mathcal{G}_{I1}}$ and $\boldsymbol{W}^r_{\mathcal{G}_{I-1}}$, we shift the original matrix at first, then we remove a part of rows in the shifted matrix since the matrices $\boldsymbol{I}_{down}$ and $\boldsymbol{I}_{up}$ have 0 element in their diagonal value. The detailed demonstration can be found in Figure 4.2b.

demonstrate that we do not need to perform matrix multiplication to get the result of $\boldsymbol{A}\boldsymbol{W}^r_{\mathcal{G}_I}$. Instead, we can simply shift and add the elements in $\boldsymbol{W}^r_{\mathcal{G}_I}$ to achieve the same goal. Since the matrix multiplication is replaced by the matrix addition, the computational cost will be very small. According to , we build the EGCN as

follow:

$$\begin{aligned} \boldsymbol{W}_{\mathcal{G}_I}^{r+1} &= \mathrm{EGCN}(\boldsymbol{W}_{\mathcal{G}_I}^{r}, \boldsymbol{A}) = \sigma\left((\boldsymbol{I}+\boldsymbol{A})\boldsymbol{W}_{\mathcal{G}_I}^{r}\boldsymbol{W}_{k}^{r}\right) \\ &= \sigma((\boldsymbol{W}_{\mathcal{G}_I}^{r} + \boldsymbol{W}_{\mathcal{G}_{IN}}^{r} + \boldsymbol{W}_{\mathcal{G}_{I1}}^{r} + \boldsymbol{W}_{\mathcal{G}_{I-N}}^{r} \\ &\quad + \boldsymbol{W}_{\mathcal{G}_{I-1}}^{r})\boldsymbol{W}_{k}^{r}), \quad r = 1, 2, \ldots, R-1. \end{aligned} \tag{4.7}$$

The EGCN adopts spectral-based graph convolution, and $\boldsymbol{W}_k^r$ represents the learnable parameters in GCN. The output of EGCN, $\boldsymbol{W}_{\mathcal{G}_I}^{R+1} \in \mathbb{R}^{N^2}$, is the importance of each node in $\mathcal{G}'$, and the edge weight between nodes, $\boldsymbol{W}\mathcal{G}'$, is calculated by using the mean of the adjacent node weights.

After that, we move to the next step, which is generating the MST according to the edge weight $\boldsymbol{W}\mathcal{G}'$. However, we find that a single MST is not stable, which means that using different random seeds during training can result in very distinct MST formulations, leading to different performance and sometimes the network cannot converge. Therefore, we propose a multi-stage MST algorithm to stabilize the training result.

4.1.3 Multi-Stage Minimum Spanning Tree

In the convolutional network [136, 48, 97, 32], different layers capture different levels of image features. For the shallow layer, context details such as image texture and image edges will be obtained, while in the deeper layer, the features are full of semantic information. Combining the features at different levels can greatly improve the network performance and make the prediction more stable [61, 69, 85, 72]. Therefore, we design an algorithm called multi-stage MST to collect MSTs from different convolution layers, and the predicted $\boldsymbol{W}\mathcal{G}'$ will be stable.

We first mark the output of ResNet in 2nd, 3rd and 4th stage as $\boldsymbol{\mathcal{X}}_m \in$

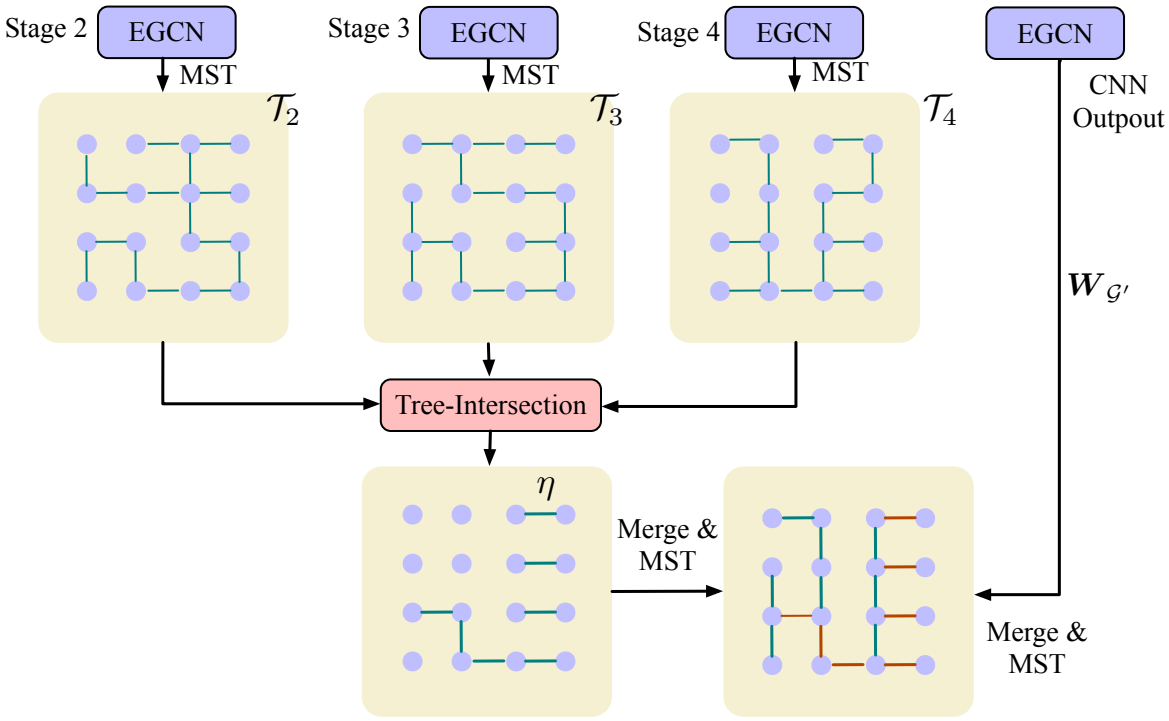


Figure 4.3: *Multi-stage MST demonstration. We first find MSTs on different ResNet outputs. Then we perform tree-intersection to produce the final MST.*

$\mathbb{R}^{C_m \times H_m \times W_m}$ and $m \in \{2,3,4\}$, they are generated by $\mathrm{F}(\boldsymbol{\mathcal{I}})$:

$$[\boldsymbol{\mathcal{X}}_2, \boldsymbol{\mathcal{X}}_3, \boldsymbol{\mathcal{X}}_4] = \mathrm{F}(\boldsymbol{\mathcal{I}}),$$

The the MST in 2nd, 3rd and 4th stage are generated and marked as τ_2, τ_3 and τ_4.

$$\tau_m = \mathcal{T}(\mathrm{EGCN}(\mathrm{L}(\boldsymbol{\mathcal{X}}_m))), \quad m \in \{2,3,4\}. \tag{4.8}$$

Then, **tree-intersection** is proposed to merge the different MST τ_1 and $\tau_2 \cdots \tau_n$. Specifically, we first obtain the adjacency matrix of each τ_n and apply element-wise multiplication to them as:

$$\tau_1 \cap \tau_2 \cdots \tau_n = \boldsymbol{A}_{\tau_1} \cdot \boldsymbol{A}_{\tau_2} \cdots \boldsymbol{A}_{\tau_n},$$

In our implementation, given τ_2, τ_3, and τ_4 that generated in Equation (4.8), the

multi-stage MST is performed as:

$$A^{M}_{W_{\mathcal{G}'}} = [1 + (\eta - 1)\tau_2 \cap \tau_3 \cap \tau_4] \cdot A_{W_{\mathcal{G}'}},$$

Here we use a small number $\eta = 1e - 3$ to control the edge weight in $A^{M}_{W_{\mathcal{G}'}}$. For an input adjacency matrix $A_{W_{\mathcal{G}'}}$ that generated by $W\mathcal{G}'$, the output of multi-stage MST will be $A^{M}_{W_{\mathcal{G}'}}$. The details of multi-stage MST can be found in Figure 4.3.

4.2 Weight Optimization

Since the SFC is obtained after the Prim algorithm [79] and the Cover-and-Merge algorithm, which are not differentiable, gradient optimizers like SGD or Adam cannot be used to optimize the model. Therefore, we propose a novel learning scheme based on the Siamese network, which will be introduced later.

4.2.1 Objectives

Since SFCs are used for data linearization, we introduce two objectives to measure the quality of the linearized feature: autocorrelation and LZW code length.

Autocorrelation. Autocorrelation is a common metric used in data transmission [70, 99] and spectral analysis [39]. It measures the correlation between the flattened feature and the k-delayed flattened feature. For example, given a sequence $\boldsymbol{Y} \in \mathbb{R}^{HW}$ obtained by image flattening, we delay it by k and get $\boldsymbol{Y}t + k]$. Autocorrelation is then calculated as:

$$\Phi_a = \frac{\sum_{i=1}^{HW-k} \boldsymbol{Y}[t]\boldsymbol{Y}[t+k]}{\sum_{i=1}^{HW} \boldsymbol{Y}[t]^2}, \tag{4.9}$$

A higher autocorrelation indicates better performance of data flattening, which implies a better SFC.

LZW Code Length. LZW Coding [144] is a lossless data compression algorithm. We measure the quality of SFC by examining the coding length after LZW coding, which is performed as:

$$\Phi_l = \text{Length}(\text{LZW Coding}(\boldsymbol{Y})). \tag{4.10}$$

Where $\boldsymbol{Y}$ is the flattened image. A shorter code length indicates better clustering ability that an SFC has, which implies a better SFC.

4.2.2 Learning Scheme

Inspired by knowledge distillation schemes [34, 88], self-learning methods [11, 28], and generative models [27, 80], we plan to use a Siamese network structure for weight optimization. Specifically, as shown in Figure 5.3, our framework has two identical branches. In the main network, the input is the original image $\mathcal{I}$, but for the Siamese network input, we add random noise to the original image ($\mathcal{I} + \mathcal{N}(0,1)$) to increase input diversity. This design will make the input image of the two networks different in context while keeping other important features intact, like edges. Both networks will produce a score calculated by the objectives that we mentioned above, and the scores of the main network and Siamese network are marked as Φ^m and Φ^s respectively. We iteratively optimize both networks following the criteria:

$$\mathcal{L}_{KD} = \begin{cases} \text{KL}\left(\boldsymbol{e}^s \| \boldsymbol{e}^m\right), & \text{If } \Phi^m \text{ is better,} \\ \text{KL}\left(\boldsymbol{e}^m \| \boldsymbol{e}^s\right), & \text{If } \Phi^s \text{ is better,} \end{cases} \tag{4.11}$$

If Φ^s performs better, we fix the parameters in the Siamese network and make it the teacher network, then we update the parameters in the main network by minimizing the KL-divergence between the main network (student) and the Siamese

network (teacher). Similarly, when Φ^m is better, we fix the parameters in the main network and optimize the Siamese network to make its predictions closer to the main network. This process is repeated every iteration during training. In the inference stage, we only use the output of the main network.

4.3 Experiments

Datasets. We test the performance of our model on the Fashion-MNIST [116], which has 60k training images and 10k testing images. Additionally, we conduct experiments on the MNIST [51] dataset. The MNIST dataset contains 70K images (60k for training and 10k for testing) composed of handwritten numbers. The original image is resized to 32×32, which is consistent with NSFC [102]. We also conduct experiments on the Tiny-ImageNet [49] dataset, which is more complex than MNIST. It is composed of 500 training, 50 validation, and 50 test images that are selected from the ImageNet dataset [18]. The original images in Tiny-ImageNet [49] have a size of 64×64, and we resize them to 32×32 during experiments. To demonstrate the scalability of our model, we conduct a Tiny-ImageNet experiment with an input size of 64×64.

Experimental Setting. The backbone of our model is the Pytorch [78] implemented ResNet-18, and we convert all images into grayscale images to accommodate the input of our model. We do not apply the pre-trained weight for the backbone, and we use $R = 3$ layers of EGCN during experiments. (R is mentioned in Equation (4.7)). We test our model after 80 epochs of training with a batch size of 128, and the learning rate is reduced by half every 20 epochs. The initial learning rate is set to $lr = 0.001$, and the Adam [41] optimizer is used to optimize the entire framework.

Dataset	Method	Autocorrelation ↑	LZW Code Length (bytes)↓
MNIST [51]	Zigzag	0.207	175.4
	Hilbert [33]	0.475	182.7 (+7.3)
	Dafner [17]	0.401	-
	NSFC [102]	0.558	171.1 (-4.3)
	Ours	**0.625**	**158.3 (-17.1)**
Fashion-MNIST [116]	Zigzag	0.552	425.8
	Hilbert [33]	0.723	427.3(+1.5)
	Dafner [17]	0.704	-
	NSFC [102]	0.786	412.4 (-13.4)
	Ours	**0.834**	**400.7 (-25.1)**
Tiny-imagenet (32×32) [49]	Zigzag	0.811	925.1
	Hilbert [33]	0.874	927.6 (+2.5)
	Dafner [17]	0.896	909.0 (-16.1)
	NSFC [102]	0.913	904.9 (-20.2)
	Ours	**0.936**	**888.7 (-36.4)**
Tiny-imagenet (64×64) [49]	Zigzag	0.719	-
	Hilbert [33]	0.773	-
	Dafner [17]	0.779	-
	NSFC [102]	-	-
	Ours	**0.826**	-

Table 4.1: *Comparison between our approach and previous methods. Here, "zigzag" represents the reshape function in PyTorch. We conduct both 32×32 and 64×64 experiments on the Tiny-ImageNet dataset to test the scalability of our model.*

4.3.1 Quantitative Comparison

Different Objectives. We present the performance of various SFCs in Table 4.1, they are evaluated on different datasets with different objectives like LZW code length

Method	GPU	Params	Inference time
GCN [43]	104M	1.6M	11ms
GAT [101]	120M	1.8M	12ms
SGC [110]	88M	0.8M	9ms
FastGCN [**fastGCN**]	96M	0.5M	8ms
EGCN	**32M**	**0.3M**	**4ms**

Table 4.2: *Inference speed and GPU occupation comparison between our model and previous works. We randomly generate a batch of* 64×64 *grid graphs as the input and the batch size is set to 512.*

Stage2	Stage3	Stage4	Autocorrelation
×	×	×	0.577 ($\pm$0.04)
✓	×	✓	0.601 ($\pm$0.02)
✓	✓	×	0.589 ($\pm$0.03)
✓	✓	✓	**0.625 (**$\pm$0.01**)**

Table 4.3: *Ablation experiment that conducted to test the multi-stage MST.*

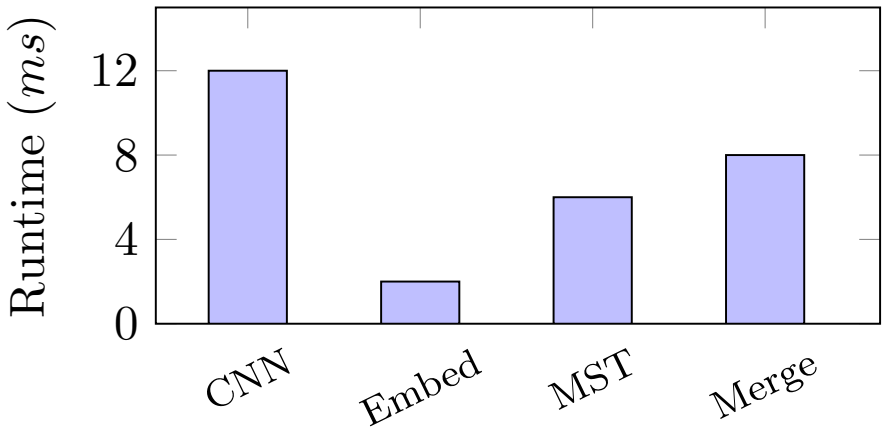


Table 4.4: *Average inference time of different components in our model.*

and autocorrelation.

We first run our model and NSFC, which are learning-based methods, and we supervise the models with a delay factor $k = 6$ for autocorrelation. Then we generate Hilbert curve, Zigzag curve, and Dafner [17] curve for comparison. The autocorrelation performance is obtained by using the mean autocorrelation of all images. After that, we measure the quality of the generated SFCs using the LZW code length metric. The experimental results are reported in Table 4.1, where we can see that our model shows the best performance in both autocorrelation and LZW code length metrics. For example, in the MNIST experiments, our model reaches 158.3 bytes in LZW code length, which is much smaller than other methods, indicating that our generated SFC has better clustering ability. Additionally, to better demonstrate the clustering ability of our SFC, we visualize some Tiny-Imagenet results in Figure 4.4.

Computational Cost. We then conduct a computational cost measurement, as our EGCN is designed as a power-saving solution for GCN. We evaluate the cost from two perspectives: the inference time of different GCN methods and their corresponding GPU memory usage. During experiments, we use a randomly generated 64×64 grid graph as the input, and we set the number of layers to 3 for all tested GCN methods. The quantitative comparison can be found in Table 4.2, where our EGCN outperforms traditional methods like GCN [101] and GAT [101] by a large margin. Also, our method is more power-saving than previous GCN-acceleration approaches like SGC [110] and FastGCN [**fastGCN**].

Since our model is a combination of different components, it is important to know the inference time of each component. Therefore, we conduct MNIST experiments to measure the running time of each part in our model, including the Cover-and-Merge algorithm, CNN backbone, graph embedding module that contains EGCN and

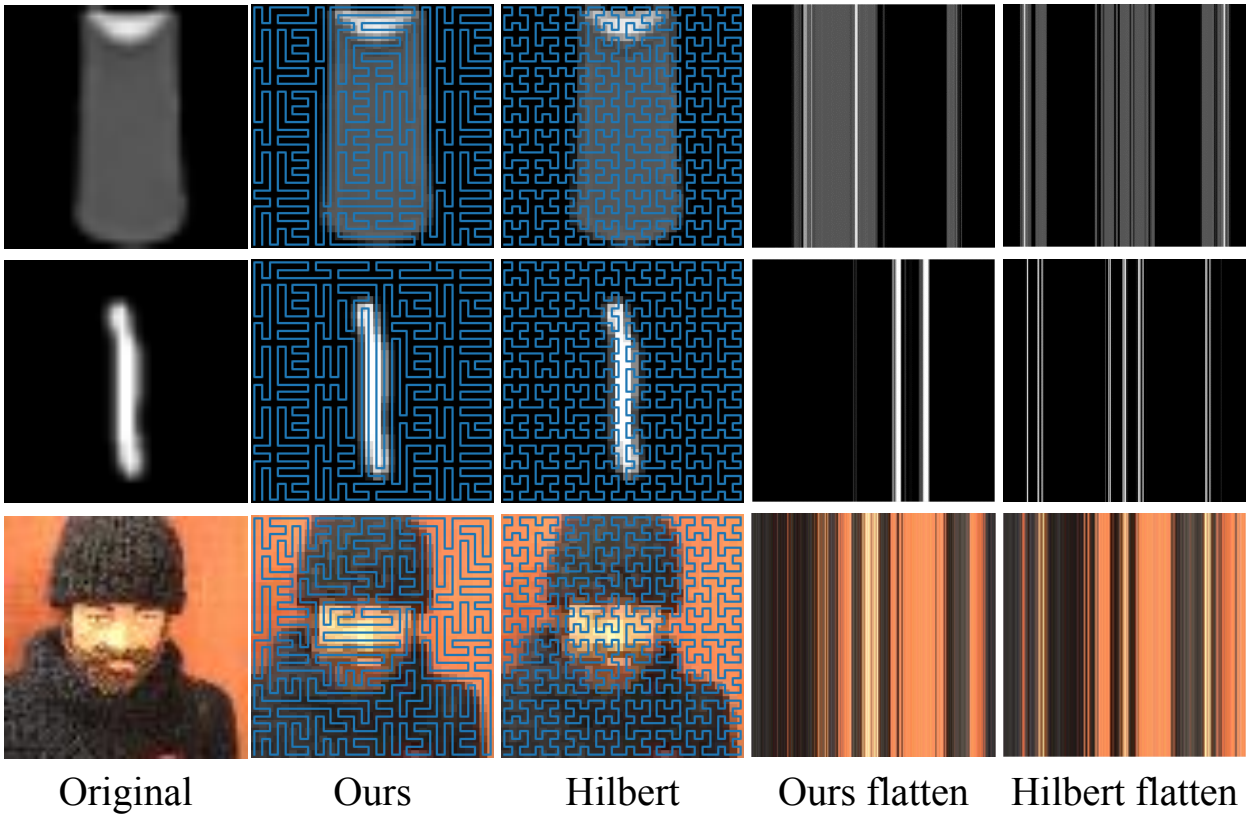


Figure 4.4: *The visualized comparison between our approach and the Hilbert curve is generated as follows: Firstly, the image is flattened to a 1D sequence using a given space-filling curve. Then, we upsample the flattened 1D sequence to a 2D image for better visualization. Compared to the Hilbert curve, our proposed method obviously has better clustering properties.*

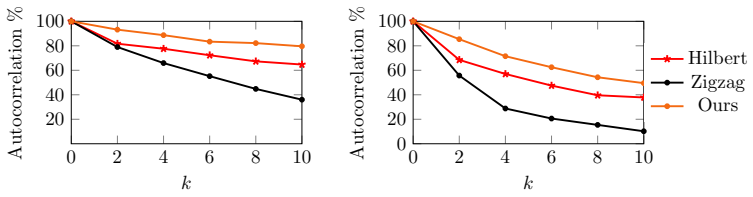
Learning Scheme	Params	Training Time/epoch	AC
NSFC [102]	22.9M	1867s	0.593
Ours	22.3M	**72s**	**0.625**

Table 4.5: *Learning scheme comparison. Params means number of parameters in the training framework.*

feature normalization, and MST generation. 4.4 records our evaluation results, in which we can see that graph embedding only consumes 2ms, further proving that our proposed EGCN is very efficient.

GCN(·)	L(·)	EGCN(·)	AC
✓	×	×	0.594
×	×	✓	0.598
✓	✓	×	0.614
×	✓	✓	**0.625**

Table 4.6: *Ablation experiments about the performance of feature normalization layer and EGCN module.*



(a) *MNIST* **(b)** *Fashion-MNIST*

Figure 4.5: *Visualized results on MNIST and Fashion-MNIST dataset when selecting different delay factor k.*

4.3.2 Ablation Study

In this part, we will test the contribution of each module in our proposed model. We use MNIST dataset with autocorrelation objective ($k = 6$) during the following ablation experiments.

Multi-Stage Minimum Spanning Tree. Multi-stage MST is designed for stable MST prediction. To evaluate the performance of this model, we first run our model without using multi-stage MST, and the result can be found in row 1 of Table 4.3. The autocorrelation is 0.577 ($\pm$0.04). When we add multi-stage MST on stage2 and stage3, the autocorrelation increases to 0.589 ($\pm$0.03). We can see that the prediction becomes more stable (from $\pm$0.04 to $\pm$0.03) and the autocorrelation becomes higher. If we apply the multi-stage MST on stage2, stage3, and stage4 of ResNet, the result will be even better. The variation of prediction becomes negligible and the autocorrelation reaches 0.625, which outperforms the model without multi-stage MST by a large margin.

Different Learning Schemes. Previous work NSFC [102] generates SFCs by first generating a proposal SFC, then using a curve evaluator to determine whether we should keep this result or not. In our research, we conduct a Siamese network-based

learning scheme and we will measure the performance of both training schemes in terms of autocorrelation and training time per epoch. To make the comparison fair, we keep all other settings (EGCN, multi-stage MST, etc.) the same except for the training scheme. 4.5 shows the result of different training schemes. It can be found that the parameters of both methods are not distinct since both methods use an additional branch to determine the quality of the generated SFC. However, our proposed method has a faster training speed since the NSFC scheme needs to apply the Dafner [17] algorithm, which is time-consuming. Additionally, our scheme shows better autocorrelation performance than the NSFC scheme, which demonstrates that our method is more suitable for SFC generation.

EGCN and Feature Normalization. In this part, we test the performance of our proposed EGCN. We simply replace all the EGCN to GCN at first, and the results can be found in 4.6. It shows that GCN performance is close to EGCN without feature normalization. This is because EGCN is just a GCN acceleration method, which will not affect the final performance. Next, we apply feature normalization to GCN and EGCN and we can find that the autocorrelation performance becomes better. It illustrates that feature normalization technique is essential to our framework.

Scalability. Scalability is a significant problem for previous works like NSFC because vanilla GCN cannot handle a large grid graph. For example, if GCN is processing a grid graph with size 256×256, the adjacency matrix of that graph will have a size of 65536×65536, which requires an unacceptable cost. Moreover, finding an MST on a large graph is still a challenging problem. However, in our model, the scalability problem is solved because EGCN is very efficient. We conducted a Tiny-Imagenet experiment to test the scalability of our model. In this dataset, all images have a size of 64×64, which is much larger than others. The autocorrelation performance in Table 4.1 shows that our model is suitable for handling large images. Note that

we do not report the performance of NSFC because the training time is too long.

More Visualization. During experiments, we use a fixed autocorrelation factor k. To better showcase the performance of different SFCs, we provide the autocorrelation results with different k values. As shown in Figure 4.5, our model consistently performs better than other space-filling curves like the Hilbert curve.

Limitations and Future Works. Searching for an optimal SFC is a challenging problem, especially in large graphs where the complexity increases significantly. Hence, we cannot ensure that our method provides the global optimal solution. As one of the future directions, we will search for more powerful and efficient optimization strategies to approximate global solutions. Another direction is that we can explore the possibility of applying the proposed SFC to other tasks, such as point cloud compression [10], data clustering [74] and *etc*.

Chapter 5

Point Cloud Analysis Using Space Filling Curve

5.1 Methodologies

As illustrated in previous parts, Space-Filling Curves (SFCs) are useful data clustering tools that can be used for data linearization. The linearized feature has a lower dimension and can be processed faster compared to its original shape. Therefore, we can apply SFCs to a segmentation framework for acceleration. In this part, we will use the Hilbert curve as an example to demonstrate how to use a space-filling curve to implement efficient segmentation in a point cloud segmentation framework.

5.1.1 Hilbert Curve Preliminaries

In the previous section, we demonstrated the analytical formulation of the Hilbert curve, which is a fractal function [91]. In this section, we will detail its characteristics. The shape of the Hilbert curve can be found in Figure 5.1.

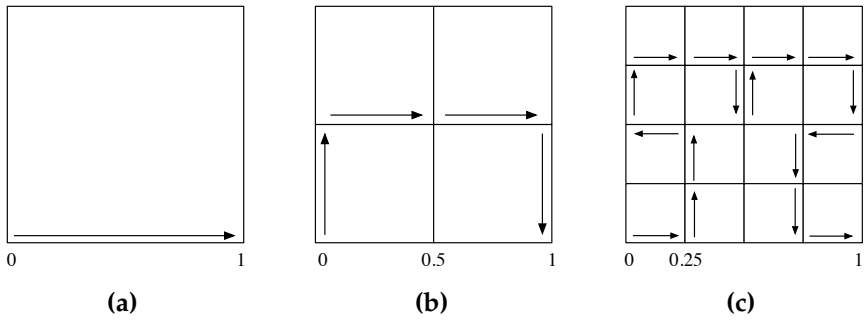
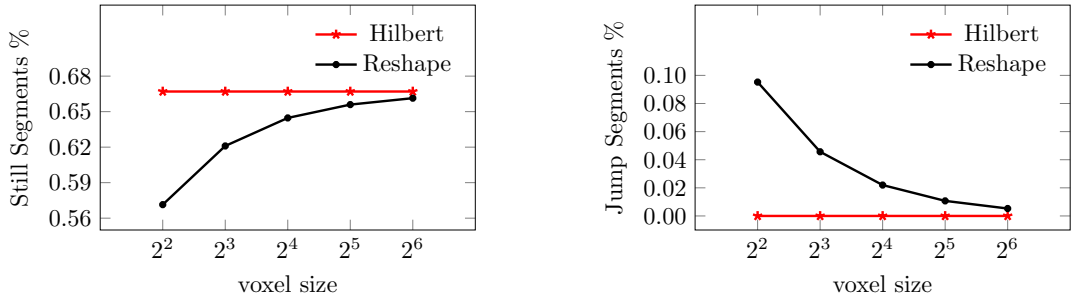


Figure 5.1: *The process of generating Hilbert curve.*



(a) *Still segments in Hilbert curve and reshape function* **(b)** *Jump segments in Hilbert curve and reshape function*

Figure 5.2: *Number of Still segments and Jump segments in Hilbert curve and reshape function. We fix the dimension $D = 3$ and increase the grid size N.*

5.1.2 Advantages of Hilbert Curve

Previously, linearization was implemented via a zigzag curve (shown in Figure 1.4), also known as the “reshape" function. Here, we provide a strong alternative solution: the Hilbert curve. Unlike the zigzag curve, the Hilbert curve has no "jump connections", implying a better locality-preserving property. Also, the distribution of the flattened feature closely resembles the original feature. To solidify our claim, we will mathematically illustrate the two advantages of the Hilbert curve in the following section.

Advantage 1: Locality Preserving.

The number of segments in the SFC [73] determines its locality-preserving ability, which is strongly correlated with its clustering ability. The better the clustering ability, the more centralized the distribution of the flattened data will be, which is more beneficial for feature extraction.

Definition 1. *Segments] A segment is defined as the "line" that links two consecutive elements. Specifically, given a grid graph with size N and dimension D, it has* $N^D - 1$ *segments that connect* N^D *points.*

Definition 2 (Jump Segments)**.** *Jump segment is defined as the segment that links two consecutive elements where the distance between them is larger than 1. For example, given two consecutive elements* P_{i+1} *and* P_i*, if* $abs(P_i - P_{i+1}) > 1$*, the segment between the two points is the Jump segment.*

Definition 3 (Still Segments)**.** *Still segment is defined as the segment that links two consecutive elements where the distance between them is equal to 0 at* k_{th} *dimension. For example, given two consecutive elements* P_{i+1} *and* P_i*, if* $abs(P_i - P_{i+1}) = 0$ *at dimension k, the segment between the two points is the Still segment.*

The Hilbert curve is known for its good locality-preserving property, which is feasible for data clustering. An intuitive comparison between the reshape function and the Hilbert curve can be found in Figure 1.4, which demonstrates that the Hilbert curve is more suitable for our task. We will then illustrate why using the Hilbert curve theoretically.

As we defined above, it is obvious that the locality-preserving property of an SFC is determined by the number of *Still* and *Jump* segments within it. The fewer jump segments, the smaller the number of jump connections, which leads to better clustering properties. Also, the more *Still* segments, the higher the spatial consistency the SFC provides, which is also good for data clustering. For a grid

graph with size N and dimension D, we can calculate the percentage of *Still* segments S_R, *Jump* segments J_R in the reshape function, and *Still* segments S_H, *Jump* segments J_H in the Hilbert curve using the following function:

$$\begin{aligned} J_R &= (\frac{N^D-1}{N-1} - D) \cdot \frac{1}{D(N^D-1)}, \\ S_R &= (DN^D - N\frac{N^D-1}{N-1}) \cdot \frac{1}{D(N^D-1)}, \\ J_H &= 0, \quad S_H = (D-1)(N^D-1) \cdot \frac{1}{D(N^D-1)}, \end{aligned} \tag{5.1}$$

The results can be found in Figure 5.2. From the figure we can find out that the Hilbert curve always has larger number of *Still* segments comapred with reshape function. Additionally, Hilbert cuvre has no *Jump* segments. Therefore, Hilbert curve is much more locality preserving than reshape function.

Advantage 2: Lower Space to Linear Ratio

Another advantage of the Hilbert curve is that the feature flattened by the Hilbert curve has a similar data distribution to its original shape. Specifically, if we apply data flattening, we will inevitably change the distribution of the original data, and some continuous points may no longer be neighbors of each other after flattening. Therefore, we should consider the similarity between the original and the flattened shape when choosing an SFC. We can check the *space to linear ratio* (SLR) of a SFC for the similarity measurement.

Definition 4 (Space to Linear Ratio)**.** *Suppose in a 2D system [0,1]×[0,1], we have two points $p(\tau)$ and $p(t)$. If mapping these two points to τ and t that in a 1D system [0,1] via a SFC p, p:[0,1]→[0,1]×[0,1], the space to linear ratio of the two points is defined as:*

$$\frac{|p(t) - p(\tau)|^2}{|t - \tau|} \tag{5.2}$$

The space to linear ratio of the SFC p is then defined as the maximum of

Equation (5.2). Clearly, a smaller SLR represents higher spatial locality as the distribution of the flattened data is closer to its original shape.

Theorem 1. *The square-to-linear ratio of the Hilbert curve is equal to 6.*

The proof is given in [4]. For the reshape function, under a certain curve order n, some consecutive elements will have an SLR of $4^n - 2^{n+1} + 2$, which is much larger than 6 as illustrated in [134]. Therefore, compared with the reshape function, the Hilbert curve is better for data flattening.

5.1.3 Pre-processing

To integrate the Hilbert curve into the segmentation model, we first introduce the Voxelization and Hilbert Flattening Module (VHFM). For all input point clouds, VHFM first voxelizes them into a 4D voxel tensor of size (C, R, R, R), consistent with previous works [83, 138, 72]. Following this, it applies slice-level data flattening as shown in Equation (5.3). In this process, each slice $\boldsymbol{\mathcal{V}}s1, \boldsymbol{\mathcal{V}}s2...\boldsymbol{\mathcal{V}}sR \in (C, R, R, 1)$ represents the corresponding feature along the Z axis.

$$\boldsymbol{\mathcal{V}}^{\in R\times R\times R} \rightarrow \begin{bmatrix} \boldsymbol{\mathcal{V}}_{s1} \\ \boldsymbol{\mathcal{V}}_{s2} \\ \vdots \\ \boldsymbol{\mathcal{V}}_{sR} \end{bmatrix} \xrightarrow{\mathcal{H}_n(s)} \begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_R \end{bmatrix} = \boldsymbol{\mathcal{I}}. \tag{5.3}$$

Next, we apply the Hilbert curve $\mathcal{H}n(\boldsymbol{s}_k) = \boldsymbol{\mathcal{V}}sk, k = 1, 2...R$ to obtain the flattened feature $\boldsymbol{s}1, \boldsymbol{s}2...\boldsymbol{s}R \in (C, R^2)$. We reconstruct the sequences along the Z-axis and the resultant feature will be $\boldsymbol{\mathcal{I}}^{\in(C,R^2,R)}$. We will use $\boldsymbol{\mathcal{I}}$ as the input feature of

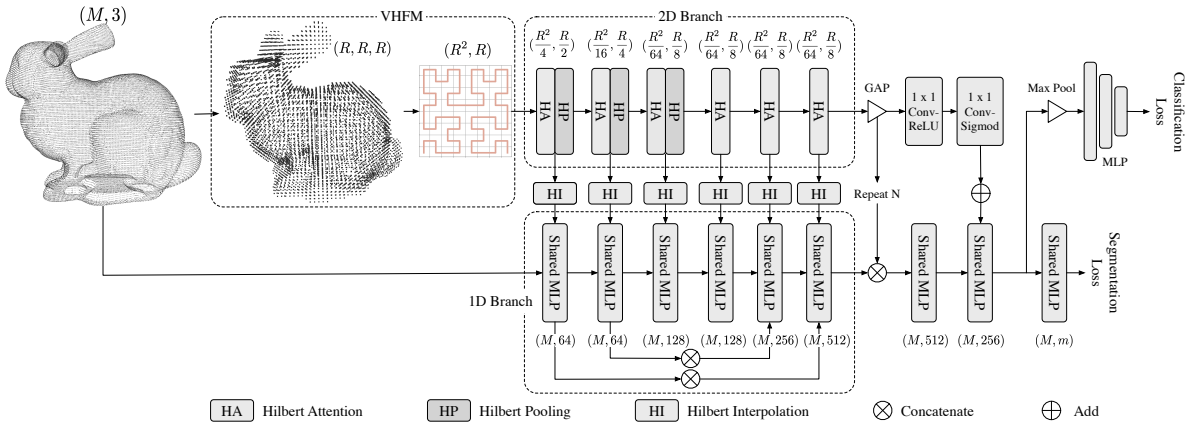


Figure 5.3: *Pipeline of our proposed HilbertNet. The input point cloud will be processed by Voxelization and Hilbert flattening module (VHFM). After that, we extract 1D point features, 2D slice features and we merge them with Hilbert Interpolation operation (HI). Channel-wise attention [37, 108] and global average pooling are also applied in our framework. Our model is designed for classification and segmentation tasks.*

the neural network. It can be observed that this is a 2D image, which uses Conv2D for feature extraction, which is much smaller than Conv3D for voxel.

We also adopt data augmentation techniques as implemented in previous works, such as point cloud rotation, point cloud flip, and random point jitting. All the data augmentations are applied directly to the point cloud, which is before VHFM.

5.1.4 HilbertNet

HilbertNet is a multiple feature fusion framework [66, 138] that combines 3D voxel features with 1D point features for accurate point cloud segmentation. However, HilbertNet is more efficient than previous works because we do not use 3D voxels directly. Instead, we initially flatten the 3D branch into a 2D image, which reduces 3D convolution to 2D and saves significant computational cost, as shown in Figure 5.3. However, this difference also implies that we cannot use previous modules and must design our own operators. Hence, we will introduce our designed operators:

Hilbert interpolation, Hilbert pooling, and Hilbert attention in details

Feature Gathering. Since the final prediction is a point cloud instead of a voxel, we need to use a “devoxelization" process that is applied to the 2D flattened voxel, transforming it into a 1D point feature. This module should have a function similar to linear interpolation [66] or attention-based interpolation [122]. However, it is not advisable to apply linear interpolation to the flattened feature since the feature obtained by first interpolating and then applying Hilbert flattening differs from the feature obtained by first applying Hilbert flattening and then interpolating. Therefore, we propose using Hilbert interpolation to replace linear interpolation.

Linear Interpolation. Recalling that the process of linear interpolation for a voxel feature $\boldsymbol{\mathcal{V}}^{\in(C,R,R,R)}$ is:

$$\boldsymbol{O} = \text{Reshape}(\boldsymbol{\mathcal{V}})\boldsymbol{F}_{linear}, \tag{5.4}$$

It transforms the 4D voxel tensor into a point cloud tensor $\boldsymbol{O}^{\in(M,C)}$ through the interpolation kernel $\boldsymbol{F}_{linear}$ and the output has M points. The linear interpolation is not locality preserving as it uses $Reshape(\cdot)$ function and the voxel to be interpolated is usually sparse, which will lower the intensity of border girds. Therefore, we propose our Hilbert interpolation.

Hilbert Interpolation. For a given 2D flattened voxel $\boldsymbol{\mathcal{I}}^{\in(C,R^2,R)}$, Hilbert interpolation $\mathfrak{L}(\cdot)$ works as:

$$\begin{aligned} &\boldsymbol{O} = \mathfrak{L}(\boldsymbol{\mathcal{I}}), \text{with} \\ &\mathcal{H}^{\lceil M \rceil}(\boldsymbol{O}) = \begin{cases} (\boldsymbol{\mathcal{I}} \cdot \boldsymbol{\mathcal{W}}_h)\boldsymbol{F}_{linear}, & \text{if } M \leq R^3; \\ \boldsymbol{\mathcal{I}}\boldsymbol{F}_{linear}, & \text{if } M > R^3. \end{cases} \end{aligned} \tag{5.5}$$

In the upsampling case ($M > R^3$), we directly use linear interpolation and the

Hilbert curve $\mathcal{H}\lceil M \rceil(\cdot)$ to transform $\boldsymbol{\mathcal{I}}$ into the point cloud feature $\boldsymbol{O}$. $\lceil M \rceil$ represents the nearest curve order where the corresponding Hilbert curve encompasses at least M points. However, in the downsampling case ($M \leq R^3$), the process becomes more complicated. We first introduce $\boldsymbol{\mathcal{W}}h$, an interpolation weight used to compensate for the border area of $\boldsymbol{\mathcal{I}}$. $\boldsymbol{\mathcal{W}}h$ is the number of empty grids in an interpolation kernel. The more empty grids there are, the larger the number in $\boldsymbol{\mathcal{W}}h$, resulting in greater compensation. Next, the tensor $\boldsymbol{\mathcal{W}}_B$ is obtained after applying a sum filter to $\boldsymbol{\mathcal{I}}_B$:

$$\boldsymbol{\mathcal{W}}_B = \boldsymbol{\mathcal{I}}_B \boldsymbol{F}_{sum}. \tag{5.6}$$

Obviously, $\boldsymbol{F}_{linear}$, which is marked as K_F, determines the size of $\boldsymbol{F}_{sum}$ and $\boldsymbol{\mathcal{W}}_B$ depends on the interpolation scale. It is easy to find that the number of non-empty grids that included in the interpolation kernel is represented by $\boldsymbol{\mathcal{W}}_B$. Finally, we use a nearest interpolation $\mathcal{N}(\cdot)$ to $\boldsymbol{\mathcal{W}}_B$ and obtain $\boldsymbol{\mathcal{W}}_h^{\in(C,R^2,R)}$ as follow:

$$\boldsymbol{\mathcal{W}}_h = K_F - \mathcal{N}(\boldsymbol{\mathcal{W}}_B) + 1. \tag{5.7}$$

The Hilbert interpolation $\mathfrak{L}(\cdot)$ transforms 2D features into 1D using Hilbert flattening and our proposed border-adaptive interpolation. In our real $\mathfrak{L}(\cdot)$ implementation, Bilinear kernel is selected. We will get the point feature $\boldsymbol{O}$ from 2D branch after the operation $\mathfrak{L}(\cdot)$. Also, the 1D branch will produce another point feature $\boldsymbol{X}$. These two point features are fused through addition to get the final output:

$$\boldsymbol{Y} = \alpha(\boldsymbol{X}) + \boldsymbol{O}, \tag{5.8}$$

Where $\alpha(\cdot)$ is a shared MLP.

Hilbert Pooling. Now let us consider the pooling module. Originally, 2D pooling compresses the spatial information of a feature map from the neighborhood of

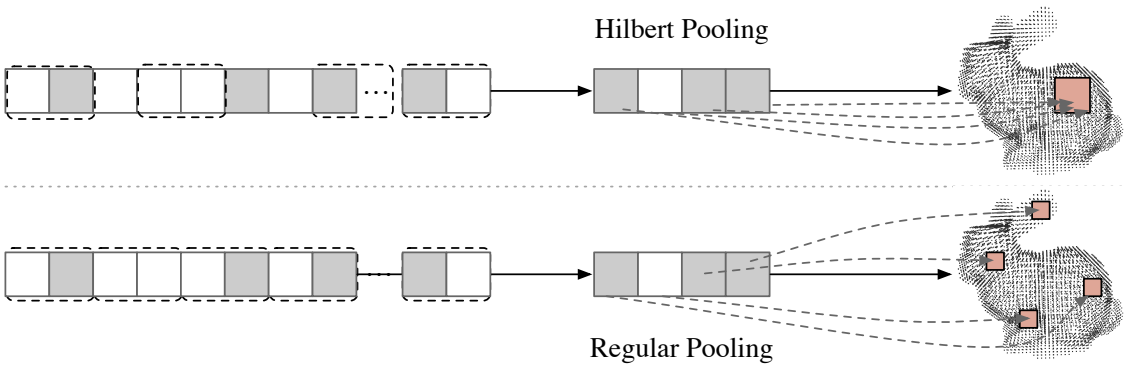


Figure 5.4: *Demonstration of Hilbert pooling and original max pooling.*

the pooling window. However, this paradigm is not applicable to the 2D feature $\boldsymbol{\mathcal{I}}$ that flattened by the Hilbert curve, as the neighboring elements in the pooling window may not be neighbors in the original 3D shape and an example can be found in Figure 5.4. In order to handle this problem and making the pooling module performs closer to the original 3D pooling, we propose Hilbert pooling. Given a tensor $\boldsymbol{\mathcal{I}}$, Hilbert pooling $\mathfrak{P}(\cdot)$ is performed as:

$$\text{MaxPool3D}(\mathcal{H}n^{-1}(\boldsymbol{\mathcal{I}})) \overset{\mathcal{H}n-1(s)}{\longrightarrow} = \boldsymbol{\mathcal{I}}' = \mathfrak{P}(\boldsymbol{\mathcal{I}}), \tag{5.9}$$

It combines original 3D pooling with inverse Hilbert curve operation $\mathcal{H}n^{-1}(\boldsymbol{\mathcal{I}})$, which is defined as the reverse of Equation (5.3), turning the 2D flattened feature into 3D. The resultant feature after Hilbert pooling will be $\boldsymbol{\mathcal{I}}'^{\in(C,\frac{R^2}{4},\frac{R}{2})}$, which is similar to Equation (5.3) since $\mathcal{H}n(s) \approx \mathcal{H}n-1(s)$.

Hilbert Attention. Similar to 2D tasks, a 2D flattened feature map has rich long-range context information, and utilizing this can greatly enhance feature extraction. The attention module [106, 128, 12] is the top priority for harvesting long-range context, and hence, we build a Hilbert attention module to collect context information within the slice (intra-slice correlation) and between slices (inter-slice correlation).

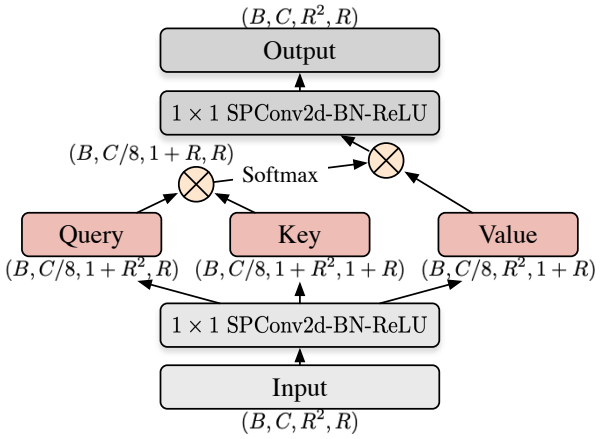


Figure 5.5: *Details of Hilbert attention. It is constructed by matrix multiplications and 2D sparse convolutions. Specifically, we generate Key and Query feature via* $\mathbf{1}\times\mathbf{4}$ *and* $\mathbf{4}\times\mathbf{1}$ *sparse 2D convolution respectively. Another* $\mathbf{4}\times\mathbf{4}$ *convolution is applied to get the value feature.*

Additionally, we calculate the mixed correlation between inter-slice and intra-slice correlations. The framework of Hilbert attention can be found in Figure 5.5.

Intra-Slice Correlation. For the intra-slice correlation measurement, we focus on the sequences $\boldsymbol{s}k$, $k \in 1, R]$, which are transformed by Equation (5.3). Specifically, we follow the paradigm in [130], using a weighted linear projection on the sequences where the weight is denoted as w_{key}. (noted as *Key* in Figure 5.5)

$$\sigma(\boldsymbol{\mathcal{I}}) = \sum_{e_k \in \boldsymbol{s}_k} w_{key} e_k \tag{5.10}$$

After that, we get the long-range dependencies of input featuremap **along** each slice.

Inter-Slice Correlation. Similarly, the inter-slice correlation can be obtained by using the transpose operation of intra-slice correlation, denoted as $\phi(\cdot)$. (noted as *Query* in Figure 5.5)

$$\phi(\boldsymbol{\mathcal{I}}) = \sigma(\boldsymbol{\mathcal{I}}^\top), \tag{5.11}$$

It collects context information **across** each slice.

Mixed Correlation. Next, we introduce mixed correlation $\gamma(\boldsymbol{\mathcal{I}})$ to measure the

importance between inter-slice and intra-slice correlations. In the code implementation, it is a 4×4 convolution. The kernel is set to 4 because the 3D Hilbert curve is a repetition of the smallest unit, a $2 \times 2 \times 2$ cell, which includes 8 elements. Therefore, a 4×4 convolution is sufficient to cover all the elements. The formulation of Hilbert attention HA is the combination of the three kinds of correlations. $HA = \text{Softmax}(\phi(\boldsymbol{\mathcal{I}})\sigma(\boldsymbol{\mathcal{I}}))\gamma(\boldsymbol{\mathcal{I}})$,

5.2 Experiments

5.2.1 Implementation Details

Experimental Setting. HilbertNet adopts the Adam optimizer [42] with a learning rate of $lr = 0.001$, consistent with previous work. In the part-segmentation and classification tasks, we set the order of the Hilbert curve to $n = 6$, which fits the voxel grid of size $64 \times 64 \times 64$. In both tasks, the batch size is 16, the learning rate decays by 50% every 50 epochs, and the training epochs are set to 200.

Since the S3DIS dataset provides larger point cloud, we enlarge the curve order to $n = 7$, which fits the grid size of $128 \times 128 \times 128$. Also, due to the GPU memory limitation, we reduce the batch size to 8 and the total training epoch is set to 80. We decay the learning rate by 50% every 20 epochs.

Classification Dataset. We evaluate the performance of our model by using the ModelNet40 [112] dataset. The dataset is comprised of 9843 objects categorized for training, and an additional 2468 objects allocated for testing. In accordance with the settings of former research [52], we consistently sample 1024 points for each experiment.

Part-Segmentation Dataset. During part-segmentation experiments, we use the ShapeNetPart dataset [7]. This dataset includes 16881 objects in total, each objects has 2 to 6 parts. These objects were divided into 16 categories. Following previous works [66], we sample 2048 point clouds during experiments.

Large Scale Segmentation Dataset. We used the S3DIS dataset [1] to assess the performance of HilbertNet in large-scale scene parsing tasks. Data from 271 rooms across 3 different buildings was collected by S3DIS. Each point in the dataset is classified into one of 13 categories. Following the previous works[122, 130], we perform experiments on Area 5.

5.2.2 Experimental Results

ModelNet40. Since ModelNet40 is a classification dataset, we append three FC layers after the main framework for point cloud classification. The results in Table 5.1 show that our model achieves top-tier performance compared to previous works. Here, we use 1024 sampling points as input, which is consistent with previous works [82, 53, 115], ensuring a fair comparison. Note that since our model collects features from both voxels and point clouds, its performance surpasses voxel-only frameworks like VoxelNet [140] and point-only frameworks like PointNet [82].

ShapeNetPart. For the segmentation task, we use a shared MLP at the end of HilbertNet to generate the output feature. The performance of the part-segmentation task can be found on the right side of Table 5.1. With the help of the 2D branch, HilbertNet shows better mIoU performance than point-only designs like SO-Net [52] and PointNet [82]. Also, the point feature is a good complement to the grid voxel feature. By using this, our model surpasses algorithms that only apply 3D or 2D

Table 5.1: *Comparison of different methods on ModelNet40 and ShapeNetPart datasets.*

ModelNet40		ShapeNetPart	
Method	Acc	Method	mIoU
VoxNet [140]	85.9	Kd-Net [45]	82.3
Subvolume [84]	89.2	PointNet [82]	83.7
PointNet [82]	89.2	SO-Net [52]	84.9
DGCNN [107]	92.9	3D-GCN [63]	85.1
PointASNL [119]	92.9	DGCNN [107]	85.2
Grid-GCN [118]	93.1	PointCNN [55]	86.1
PCT [29]	93.2	PVCNN [66]	86.2
SO-Net [52]	93.4	KPConv [98]	86.4
CurveNet [115]	93.8	CurveNet [115]	86.6
Ours	**94.1**	Ours	**87.1**

Table 5.2: *Inference time and mIoU comparison between HilbertNet and other approaches.*

Method	Inference time	voxel size	mIoU
3D-UNet[72]	347ms	64^3	84.2
PVCNN[66]	62.5ms	32^3	86.0
HilbertNet-L	**42.1ms**	64^3	85.8
HilbertNet-M	59.2ms	64^3	86.4
HilbertNet	91.6ms	64^3	**87.1**

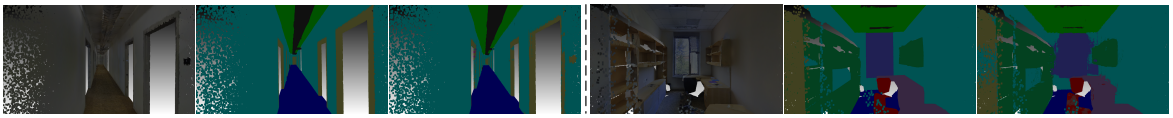
Table 5.3: *Comparison between Hilbert Attention and other convolution modules. We use a randomly generated 32^3 voxel as the input. FLOPs is the short of floating point operations.*

Method	FLOPs	GPU Memory
3D Convolution	18.86G	162M
2D Convolution	4.45G	148.7M
Sparse 2D Convolution	1.47G	49.6M
NonLocal	0.34G	4G
Hilbert Attention	**0.32G**	**47.8M**

features, like 3D-GCN [63].

S3DIS. Since HilbertNet uses a voxel grid, a frequently asked question is whether it can handle large point clouds. To answer this question, we conducted an experiment using the S3DIS [1] dataset. We used the Area 5 set, and the experimental results in Table 5.4 demonstrate that HilbertNet is capable of handling large-scale point clouds. The Hilbert curve significantly reduces the computational cost for 3D feature extraction, and with the aid of Hilbert attention, our model surpasses most state-of-the-art frameworks like pointTransformer [130], which is a transformer-based model. Additionally, we have visualized some results generated by HilbertNet in Figure 5.6a and some results generated by other methods in Figure 5.6b, providing a straightforward way to appreciate the effectiveness of HilbertNet.

Inference Speed. Inference speed is a crucial aspect of this study, as the goal



(a) *Left to right: Point Cloud, GT, HilbertNet.*

(b) *Left to right: Point Cloud, GT, PointNet[82], KPConv[98], HilbertNet.*

Figure 5.6: *(a) Visualized performance of HilbertNet on S3DIS Area 5; (b) Comparison between HilbertNet and other methods.*

Table 5.4: *Performance of different approaches on S3DIS Area 5 dataset.*

	PointTransformer [130]	MinkowskiNet [14]	PointCNN [55]	KPConv [98]	PCCN [105]	PointNet [82]	HilbertNet
column	38	34.1	17.6	23.9	6	3.9	**37.6**
window	63.4	48.9	22.8	58	**69.5**	46.3	64.1
chair	82.4	89.8	80.6	**91**	65.6	52.6	85.4
clutter	59.3	58.6	56.7	58.9	46.2	33.2	**60.1**
wall	86.3	86.2	79.4	82.4	75.9	69.8	**88.9**
door	**74.3**	62.4	62.1	69	63.5	10.8	73.8
table	**89.1**	81.6	74.4	81.5	66.9	59	88.4
sofa	74.3	47.2	31.7	**75.4**	47.3	5.9	73.5
bookcase	80.2	74.9	66.7	75.3	68.9	40.3	**82.7**
board	**76**	74.4	62.1	66.7	59.1	26.4	74.7
beam	0	0	**0.3**	0	0	0.1	0
floor	98.5	**98.7**	98.2	97.3	96.2	97.3	97.8
ceiling	94	91.8	92.3	92.8	92.3	88.8	**94.6**
mIoU	70.4	65.4	57.3	67.1	58.3	41.1	**70.9**

of applying the Hilbert curve to the segmentation framework is to accelerate it. Therefore, we should compare the computational overhead of our methods with models that have a similar pipeline, like PVCNN [66]. In addition to the original HilbertNet, we propose two variations named HilbertNet-M and HilbertNet-L for

different application scenarios. HilbertNet-M reduces the channel number of the original HilbertNet by half while keeping other settings intact. HilbertNet-L only has 0.25× channels and is designed for low-latency applications. We tested all the models, including HilbertNet, PVCNN, and 3D-UNet [72] (a baseline model), on the ShapeNet dataset with a GTX TITANX GPU. All the tested models were reproduced according to their original papers. The comparison of different models can be found in Table 5.2. It can be observed that HilbertNet-L has the highest inference speed with a comparably high mIoU, while the original HilbertNet delivers the best performance with an acceptable inference speed.

Next, we delve into the details of HilbertNet, examining the inference speed of different convolution modules. Table 5.3 shows the FLOPs and GPU usage of different convolution methods, including vanilla 2D and 3D convolution, vanilla Non-local attention, sparse 2D convolution, and Hilbert Attention. The results show that Hilbert Attention outperforms all the listed modules in terms of FLOPs and GPU memory usage. This outcome explains why HilbertNet is efficient and demonstrates the power of the Hilbert curve in accelerating the segmentation framework.

5.2.3 Ablation Study

In this section, we will use the ModelNet40 dataset to conduct an ablation study and determine the contribution of each part. We evaluated different feature flattening strategies, interpolation algorithms, and pooling and convolution methods.

Hilbert Curve vs. Reshape Function. Here, we simply replace all instances of the Hilbert curve in HilbertNet with the reshape function, and the performance can be found in Figure 5.7. The results show that when we substitute the Hilbert curve with the reshape function, the accuracy drops from 94.1% to 91.2%. This is

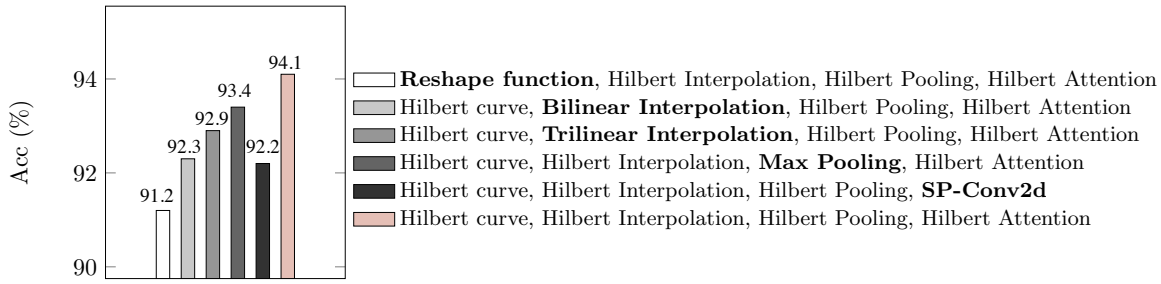


Figure 5.7: *Ablation experiments of different modules in HilbertNet, including* ***Convolution Method, Pooling Method, Flattening Method, Gathering Method****.*

consistent with the fact that the Hilbert curve has a much better locality-preserving ability. Also, the figure suggests that the Hilbert curve contributes significantly to the improvement in accuracy.

Hilbert Interpolation vs. Linear Interpolation. Next, we examine the 2nd, 3rd, and 6th columns in Figure 5.7, where the results are generated using different interpolation methods including Bilinear, Trilinear (see Equation (5.4)), and Hilbert interpolation. Hilbert interpolation results in the highest accuracy, and Trilinear achieves the second-highest performance. This implies that Hilbert interpolation not only outperforms Bilinear interpolation but is also superior to Trilinear interpolation. This result further substantiates our claim that the Hilbert curve is locality-preserving.

Hilbert Pooling vs. 2D Max Pooling. Now let us examine the pooling module. Since we use a 2D flattened voxel, we should compare our Hilbert pooling module with traditional 2D max pooling, which is widely used in 2D tasks [32, 95, 97]. The results in the 4th and 6th columns of Figure 5.7 suggest that when processing 2D Hilbert-flattened voxels, Hilbert pooling is the optimal choice.

Hilbert Attention vs. 2D Convolution. Here, we use a 2D sparse convolution

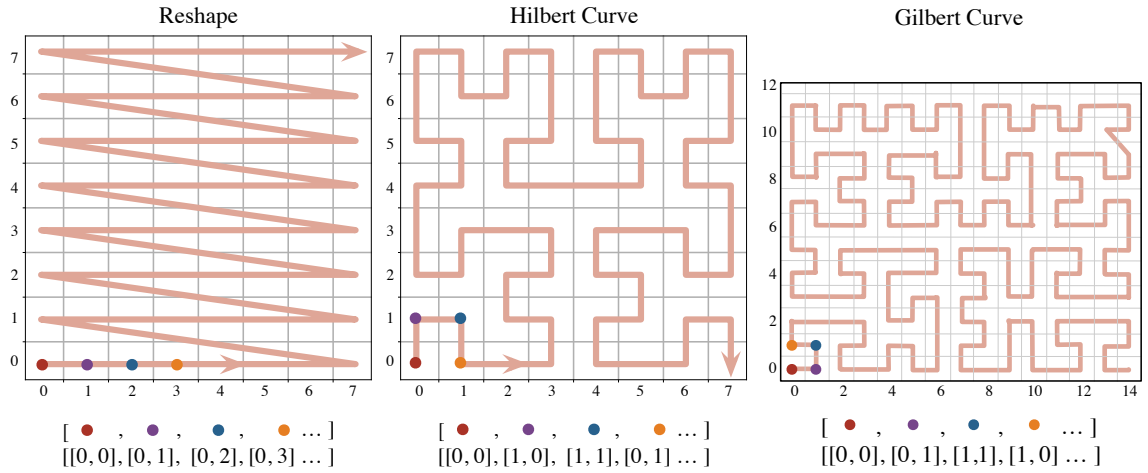


Figure 5.8: *Left: Reshape function. Middle:Hilbert curve Right:Gilbert curve. Gilbert curve is the generalized formulation of Hilbert curve, which can fit arbitrary size of input. However, the jump connections are introduced in Gilbert curve, which may reduce its clustering property.*

with a kernel size of 4×4 to replace Hilbert attention. The performance of this design can be found in the 5th column of Figure 5.7. The significant reduction in prediction accuracy indicates that without Hilbert attention, the network is incapable of harvesting context information from the 2D flattened voxel.

Limitation and Future Works. Although the Hilbert curve is an ideal solution for data clustering, its application is limited by its shape as it can only be used for voxels with a side length of 2^n, such as 16, 32, 64, and we should scale the given voxels to 16^3, 32^3, and 64^3 before using the Hilbert curve. In point cloud analysis, this setting may cause inconvenience. Firstly, the real-world voxel data is usually very large, and interpolating it is a time-consuming operation. Also, the scale function will change the structure of the original data, which will lead to incorrect segmentation. To address this limitation of the Hilbert curve, we further explore the Gilbert curve, i.e., the generalized Hilbert curve, which adapts to any side length of point clouds by adding a small part of jump connections. The comparison between the Hilbert curve and the Gilbert curve can be found in Figure 5.8. These added connections

preserve spatial locality and geometry in a more flexible manner.

Therefore, in future work, in addition to HilbertNet, we will also propose GilbertNet, which uses the Gilbert curve [127] to flatten voxels. GilbertNet sacrifices a portion of locality preservation (by introducing some jump connections) to accommodate arbitrary-shaped voxel inputs, whereas HilbertNet can only accept input voxels with side lengths of 2^n, such as 16, 32, and 64. GilbertNet offers broader applicability, and HilbertNet can be considered a special case of GilbertNet.

After that, we will explore the possibility of using data-adaptive space-filling curves in segmentation models. However, data-adaptive SFC generation is usually computationally heavy because minimum spanning tree searching, which is a key part of data-adaptive SFC generation, and hence an efficient MST searching algorithm should be proposed first.

Chapter 6

Conclusion

Our research aims to speed up deep-learning-based visual segmentation while maintaining the high performance of the segmentation model. To achieve this goal, we are focusing on optimizing the attention block, which is a key component in segmentation networks and requires a lot of computational resources. After obtaining efficient attention via tensor low-rank reconstruction, we found that the attention slimming strategy has its own limitations. In other words, we cannot infinitely reduce the size of the attention block to speed up neural networks while keeping its performance good. Therefore, we have to look for a new way to further accelerate the segmentation process and we found that space-filling curves are useful to our research since they can linearize data from higher dimensions to lower dimensions, which is similar to lossless data compression. Specifically, we can utilize the data that is compressed by space-filling curves to achieve lower cost. But before that, we have to know what kind of SFC we need and what the properties of different SFCs are. Therefore, we conduct research focusing on SFC generation and find out that the Hilbert curve is suitable for us to achieve low-cost segmentation. Finally, we combine the Hilbert curve with low-rank attention

in the point cloud segmentation task and achieve SOTA performance with very low computational overhead. The thesis is organized into 3 sections, each section reports one of our explorations including efficient attention via tensor low-rank reconstruction, study in space-filling curve generation, and applying the Hilbert curve to visual segmentation.

In the section 3, we reported how to use tensor CP reconstruction to model the attention block with a bunch of rank-1 tensors. Different from previous works that generate attention from a 2D similarity matrix, our attention is built in 3D directly. We propose a tensor generation module (TGM) to collect fragments of context information in the feature map and then use our designed tensor reconstruction module (TRM) to merge all the context fragments that come from TGM. Our attention contains spatial and channel context information simultaneously, which is different from previous works that only contain context information in spatial or channel. Additionally, our attention is 100$\times$ faster than previous works.

In the section 4, we conducted an in-depth study of the properties of different space-filling curves. Specifically, we explored the properties and applications of space-filling curves and found out that the Hilbert curve can be used for data linearization and can further accelerate the segmentation process. In order to have a deeper understanding of SFC, we propose a deep-learning-based SFC and compare our results with the Hilbert curve. During our SFC generation research, we find that SFC generation is a Hamiltonian path finding process, which requires a graph neural network (GNN). However, the vanilla GNN is not applicable to image grid graphs and therefore, we propose our efficient-GCN (EGCN), which does not need to perform matrix multiplication and is hence much faster than GNN. Additionally, we proposed a Siamese network learning scheme to optimize our SFC generation framework efficiently.

In the section 5, based on the findings in the previous two sections, we perform efficient segmentation using the Hilbert curve and low-rank attention. Specifically, we use point cloud segmentation as an example and the segmentation framework is named as HilbertNet. In the pre-processing stage of HilbertNet, we propose a Voxelization and Hilbert Flattening Module (VHFM), which first converts point cloud data into 3D voxels and then uses the Hilbert curve to flatten them into 2D. Since 2D convolution is much more power-saving than 3D convolution, the goal of efficient segmentation is achieved. Additionally, since the data distribution of flattened voxels is different from traditional 2D images, we designed several blocks like Hilbert interpolation, Hilbert pooling, and Hilbert attention to accommodate these differences. Our proposed Hilbert attention is a low-rank attention and has a lower computational cost compared to the vanilla attention block. With the help of these blocks and the Hilbert curve, our HilbertNet achieves top-1 performance in various datasets with relatively low cost.

References

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3d semantic parsing of large-scale indoor spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1534–1543.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A deep convolutional encoder-decoder architecture for image segmentation". In: 39.12 (2017), pp. 2481–2495.

[3] Zoran Balkić, Damir Šoštarić, and Goran Horvat. "GeoHash and UUID identifier for multi-agent systems". In: *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. Springer. 2012, pp. 290–298.

[4] Konstantin Evgen'evich Bauman. "The dilation factor of the Peano-Hilbert curve". In: *Mathematical Notes* 80.5 (2006), pp. 609–620.

[5] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*. Vol. 290. Macmillan London, 1976.

[6] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. "COCO-Stuff: Thing and Stuff Classes in Context". In: 2018, pp. 1209–1218.

[7] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[8] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[9] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: 2018, pp. 801–818.

[10] Wanli Chen, Xinge Zhu, Guojin Chen, and Bei Yu. "Efficient Point Cloud Analysis Using Hilbert Curve". In: ().

[11] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[12] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. "Aˆ 2-Nets: Double Attention Networks". In: 2018, pp. 352–361.

[13] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. "Attention-based models for speech recognition". In: 2015, pp. 577–585.

[14] Christopher Choy, JunYoung Gwak, and Silvio Savarese. "4d spatio-temporal convnets: Minkowski convolutional neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084.

[15] Peishan Cong, Xinge Zhu, and Yuexin Ma. “Input-Output Balanced Framework for Long-Tailed Lidar Semantic Segmentation”. In: *2021 IEEE International Conference on Multimedia and Expo (ICME)* (2021), pp. 1–6.

[16] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. “Attention-over-attention neural networks for reading comprehension”. In: 2017.

[17] Revital Dafner, Daniel Cohen-Or, and Yossi Matias. “Context-based space filling curves”. In: *Computer Graphics Forum*. Vol. 19. 3. Wiley Online Library. 2000, pp. 209–218.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A large-scale hierarchical image database”. In: 2009, pp. 248–255.

[19] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. “Learning heuristics for the tsp by policy gradient”. In: *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer. 2018, pp. 170–181.

[20] Henghui Ding, Xudong Jiang, Bing Shuai, Ai Qun Liu, and Gang Wang. “Semantic Correlation Promoted Shape-Variant Context for Segmentation”. In: 2019, pp. 8885–8894.

[21] Henghui Ding, Xudong Jiang, Bing Shuai, Ai Qun Liu, and Gang Wang. “Context Contrasted Feature and Gated Multi-Scale Aggregation for Scene Segmentation”. In: 2018, pp. 2393–2402.

[22] Carlos Esteves, Yinshuang Xu, Christine Allen-Blanchette, and Kostas Daniilidis. “Equivariant multi-view networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1568–1577.

[23] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. “The pascal visual object classes (VOC) challenge”. In: 88.2 (2010), pp. 303–338.

[24] Christos Faloutsos. “Multiattribute hashing using gray codes”. In: *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*. 1986, pp. 227–238.

[25] Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. “GVCNN: Group-view convolutional neural networks for 3D shape recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 264–272.

[26] Jun Fu, Jing Liu, Haijie Tian, Zhiwei Fang, and Hanqing Lu. “Dual attention network for scene segmentation”. In: *arXiv preprint arXiv:1809.02983* (2018).

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[28] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[29] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. “PCT: Point cloud transformer”. In: *arXiv preprint arXiv:2012.09688* (2020).

[30] Junjun He, Zhongying Deng, and Yu Qiao. “Dynamic Multi-Scale Filters for Semantic Segmentation”. In: 2019, pp. 3562–3572.

[31] Junjun He, Zhongying Deng, Lei Zhou, Yali Wang, and Yu Qiao. “Adaptive Pyramid Context Network for Semantic Segmentation”. In: 2019, pp. 7519–7528.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: 2016, pp. 770–778.

[33] David Hilbert. “Über die stetige Abbildung einer Linie auf ein Flächenstück”. In: *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes*. Springer, 1935, pp. 1–2.

[34] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[35] Yuenan Hou, Xinge Zhu, Yuexin Ma, Chen Change Loy, and Yikang Li. “Point-to-Voxel Knowledge Distillation for LiDAR Semantic Segmentation”. In: *ArXiv* abs/2206.02099 (2022).

[36] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. “Lora: Low-rank adaptation of large language models”. In: *arXiv preprint arXiv:2106.09685* (2021).

[37] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: 2018, pp. 7132–7141.

[38] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. “CCNet: Criss-Cross Attention for Semantic Segmentation”. In: 2019, pp. 603–612.

[39] Bogdan Kasztenny. “A new method for fast frequency measurement for protection applications”. In: *13th International Conference on Developments in Power System Protection*. 2016.

[40] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. “Learning combinatorial optimization algorithms over graphs”. In: *Advances in neural information processing systems* 30 (2017).

[41] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[42] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: 2015.

[43] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[44] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. “Segment anything”. In: *arXiv preprint arXiv:2304.02643* (2023).

[45] Roman Klokov and Victor Lempitsky. “Escape from cells: Deep kd-networks for the recognition of 3d point cloud models”. In: *ICCV*. 2017, pp. 863–872.

[46] Tamara G Kolda and Brett W Bader. “Tensor decompositions and applications”. In: 51.3 (2009).

[47] Wouter Kool, Herke Van Hoof, and Max Welling. “Attention, learn to solve routing problems!” In: *arXiv preprint arXiv:1803.08475* (2018).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: 2012, pp. 1097–1105.

[49] Ya Le and Xuan Yang. “Tiny imagenet visual recognition challenge”. In: *CS 231N* 7.7 (2015), p. 3.

[50] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. “Speeding-up convolutional neural networks using finetuned CP-decomposition”. In: 2015.

[51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[52] Jiaxin Li, Ben M Chen, and Gim Hee Lee. “SO-Net: Self-Organizing Network for Point Cloud Analysis”. In: 2018.

[53] Jiaxin Li, Ben M Chen, and Gim Hee Lee. “So-net: Self-organizing network for point cloud analysis”. In: *CVPR*. 2018, pp. 9397–9406.

[54] Xia Li, Zhisheng Zhong, Jianlong Wu, Yibo Yang, Zhouchen Lin, and Hong Liu. “Expectation-Maximization Attention Networks for Semantic Segmentation”. In: 2019, pp. 9167–9176.

[55] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. “PointCNN: Convolution On X-Transformed Points”. In: 2018, pp. 820–830.

[56] Jan-Yie Liang, Chih-Sheng Chen, Chua-Huang Huang, and Li Liu. “Lossless compression of medical images using Hilbert space-filling curves”. In: *Computerized Medical Imaging and Graphics* 32.3 (2008), pp. 174–182.

[57] Xiaodan Liang, Eric Xing, and Hongfei Zhou. “Dynamic-Structured Semantic Propagation Network”. In: 2018, pp. 752–761.

[58] Di Lin, Yuanfeng Ji, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. "Multi-scale context intertwining for semantic segmentation". In: 2018, pp. 603–619.

[59] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation". In: (2017), pp. 1925–1934.

[60] Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid. "Efficient piecewise training of deep structured models for semantic segmentation". In: 2016, pp. 3194–3203.

[61] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection". In: 2017, pp. 2117–2125.

[62] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft COCO: Common objects in context". In: 2014, pp. 740–755.

[63] Zhi-Hao Lin, Sheng-Yu Huang, and Yu-Chiang Frank Wang. "Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis". In: *CVPR*. 2020, pp. 1800–1809.

[64] Ce Liu, Jenny Yuen, and Antonio Torralba. "SIFT Flow: Dense Correspondence across Scenes and Its Applications". In: 33.5 (2011), pp. 978–994.

[65] Wei Liu, Andrew Rabinovich, and Alexander C Berg. "ParseNet: Looking wider to see better". In: *arXiv preprint arXiv:1506.04579* (2015).

[66] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. "Point-voxel cnn for efficient 3d deep learning". In: *arXiv preprint arXiv:1907.03739* (2019).

[67] Ziwei Liu, Xiaoxiao Li, Ping Luo, Chen-Change Loy, and Xiaoou Tang. "Semantic image segmentation via deep parsing network". In: 2015, pp. 1377–1385.

[68] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: 2015, pp. 3431–3440.

[69] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[70] Yossi Matias and Adi Shamir. "A video scrambling technique based on space filling curves". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 398–417.

[71] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. "Rangenet++: Fast and accurate lidar semantic segmentation". In: *IROS*. IEEE. 2019, pp. 4213–4220.

[72] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation". In: *2016 fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 565–571.

[73] Mohamed F Mokbel, Walid G Aref, and Ibrahim Kamel. "Analysis of multi-dimensional space-filling curves". In: *GeoInformatica* 7.3 (2003), pp. 179–209.

[74] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. "Analysis of the clustering properties of the Hilbert space-filling curve". In: *IEEE Transactions on knowledge and data engineering* 13.1 (2001), pp. 124–141.

[75] Jack A Orenstein. “Spatial query processing in an object-oriented database system”. In: *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*. 1986, pp. 326–336.

[76] Tarek Ouni, Arij Lassoued, and Mohamed Abid. “Gradient-based Space Filling Curves: Application to lossless image compression”. In: *2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*. IEEE. 2011, pp. 437–442.

[77] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. “Automatic differentiation in PyTorch”. In: *NIPS Workshop*. 2017.

[78] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[79] Robert Clay Prim. “Shortest connection networks and some generalizations”. In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401.

[80] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. “Variational autoencoder for deep learning of images, labels and captions”. In: *Advances in neural information processing systems* 29 (2016).

[81] Albert Pumarola, Stefan Popov, Francesc Moreno-Noguer, and Vittorio Ferrari. “C-flow: Conditional generative flow models for images and 3d point clouds”. In: *CVPR*. 2020, pp. 7949–7958.

[82] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *CVPR*. 2017, pp. 652–660.

[83] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. “Volumetric and multi-view cnns for object classification on 3d data”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5648–5656.

[84] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. “Volumetric and multi-view cnns for object classification on 3d data”. In: *CVPR*. 2016, pp. 5648–5656.

[85] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *arXiv preprint arXiv:1706.02413* (2017).

[86] Dian Qin, Haishuai Wang, Zhe Liu, Hongjia Xu, Sheng Zhou, and Jiajun Bu. “Hilbert Distillation for Cross-Dimensionality Networks”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 11726–11738.

[87] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. “Learning transferable visual models from natural language supervision”. In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[88] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. “Fitnets: Hints for thin deep nets”. In: *arXiv preprint arXiv:1412.6550* (2014).

[89] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: 2015, pp. 234–241.

[90] Zhang Rui, Tang Sheng, Yongdong Zhang, Jintao Li, and Shuicheng Yan. “Scale-Adaptive Convolutions for Scene Parsing”. In: 2017, pp. 2031–2039.

[91] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[92] Hanan Samet. *Applications of spatial data structures: computer graphics, image processing, and GIS*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[93] Abhishek Sharma, Oncel Tuzel, and Ming-Yu Liu. “Recursive context propagation network for semantic scene labeling”. In: 2014.

[94] Bing Shuai, Zhen Zup, Bing Wang, and Gang Wang. “Scene Segmentation with DAG-Recurrent Neural Networks”. In: 40.6 (2018), pp. 1480–1493.

[95] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: 2015, pp. 1–14.

[96] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. “Multi-view convolutional neural networks for 3d shape recognition”. In: *ICCV*. 2015, pp. 945–953.

[97] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In: 2015, pp. 1–9.

[98] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. “Kpconv: Flexible and deformable convolution for point clouds”. In: *ICCV*. 2019, pp. 6411–6420.

[99] Jan Van Sickle. *GPS for land surveyors*. CRC press, 2008.

[100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: 2017, pp. 5998–6008.

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

[102] Hanyu Wang, Kamal Gupta, Larry Davis, and Abhinav Shrivastava. "Neural Space-filling Curves". In: *arXiv preprint arXiv:2204.08453* (2022).

[103] Jun Wang and Jie Shan. "Space filling curve based point clouds index". In: *Proceedings of the 8th International Conference on GeoComputation*. Citeseer. 2005, pp. 551–562.

[104] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. "O-cnn: Octree-based convolutional neural networks for 3d shape analysis". In: *ACM Transactions On Graphics (TOG)* 36.4 (2017), pp. 1–11.

[105] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. "Deep parametric continuous convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2589–2597.

[106] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. "Non-local neural networks". In: 2018, pp. 7794–7803.

[107] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. "Dynamic graph cnn for learning on point clouds". In: *TOG* 38.5 (2019), pp. 1–12.

[108] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. "CBAM: Convolutional block attention module". In: 2018, pp. 3–19.

[109] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. “Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud”. In: *ICRA*. IEEE. 2018, pp. 1887–1893.

[110] F. Wu, T. Zhang, Ahd Souza, C. Fifty, T. Yu, and K. Q. Weinberger. “Simplifying Graph Convolutional Networks”. In: (2019).

[111] Wenxuan Wu, Zhongang Qi, and Li Fuxin. “Pointconv: Deep convolutional networks on 3d point clouds”. In: *CVPR*. 2019, pp. 9621–9630.

[112] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. “3d shapenets: A deep representation for volumetric shapes”. In: *CVPR*. 2015, pp. 1912–1920.

[113] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. “Bridging category-level and instance-level semantic image segmentation”. In: *arXiv preprint arXiv:1605.06885* (2016).

[114] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. “Wider or deeper: Revisiting the resnet model for visual recognition”. In: 90 (2019), pp. 119–133.

[115] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. “Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis”. In: *arXiv preprint arXiv:2105.01288* (2021).

[116] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[117] Chenfeng Xu, Shijia Yang, Bohan Zhai, Bichen Wu, Xiangyu Yue, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. “Image2Point: 3D Point-Cloud Understanding with Pretrained 2D ConvNets”. In: *arXiv preprint arXiv:2106.04180* (2021).

[118] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. “Grid-gcn for fast and scalable point cloud learning”. In: *CVPR*. 2020, pp. 5661–5670.

[119] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. “Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling”. In: *CVPR*. 2020, pp. 5589–5598.

[120] Jimei Yang, Brian Price, Scott Cohen, and Ming Hsuan Yang. “Context Driven Scene Parsing with Attention to Rare Classes”. In: 2014, pp. 3294–3301.

[121] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. “Hierarchical attention networks for document classification”. In: 2016, pp. 1480–1489.

[122] Maosheng Ye, Shuangjie Xu, Tongyi Cao, and Qifeng Chen. “Drinet: A dual-representation iterative learning network for point cloud segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7447–7456.

[123] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. “Learning a discriminative feature network for semantic segmentation”. In: 2018, pp. 1857–1866.

[124] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. “On compressing deep models by low rank and sparse decomposition”. In: 2017, pp. 7370–7379.

[125] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. “Context encoding for semantic segmentation”. In: 2018, pp. 7151–7160.

[126] Hang Zhang, Han Zhang, Chenguang Wang, and Junyuan Xie. “Co-Occurrent Features in Semantic Segmentation”. In: 2019, pp. 548–557.

[127] Jian Zhang, Sei-ichiro Kamata, and Yoshifumi Ueshige. “A pseudo-hilbert scan algorithm for arbitrarily-sized rectangle region”. In: *Advances in Machine Vision, Image Processing, and Pattern Analysis: International Workshop on Intelligent Computing in Pattern Analysis/Synthesis, IWICPAS 2006 Xi'an, China, August 26-27, 2006 Proceedings*. Springer. 2006, pp. 290–299.

[128] Songyang Zhang, Xuming He, and Shipeng Yan. “LatentGNN: Learning Efficient Non-local Relations for Visual Recognition”. In: 2019, pp. 7374–7383.

[129] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. “Polarnet: An improved grid representation for online lidar point clouds semantic segmentation”. In: *CVPR*. 2020, pp. 9601–9610.

[130] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. “Point transformer”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16259–16268.

[131] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid scene parsing network”. In: 2017, pp. 2881–2890.

[132] Hengshuang Zhao, Yi Zhang, Shu Liu, Jianping Shi, Chen Change Loy, Dahua Lin, and Jiaya Jia. “PSANet: Point-wise spatial attention network for scene parsing”. In: 2018, pp. 267–283.

[133] Lin Zhao, Hui Zhou, Xinge Zhu, Xiao Song, Hongsheng Li, and Wenbing Tao. “LIF-Seg: LiDAR and Camera Image Fusion for 3D LiDAR Semantic Segmentation”. In: *ArXiv* abs/2108.07511 (2021).

[134] Qingsong Zhao, Zhipeng Zhou, Shuguang Dou, Yangguang Li, Rui Lu, Yin Wang, and Cairong Zhao. “Rethinking the Zigzag Flattening for Image Reading”. In: *arXiv preprint arXiv:2202.10240* (2022).

[135] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. “Conditional random fields as recurrent neural networks”. In: 2015, pp. 1529–1537.

[136] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

[137] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. “Scene parsing through ADE20K dataset”. In: 2017, pp. 633–641.

[138] Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. “Cylinder3d: An effective 3d framework for driving-scene lidar semantic segmentation”. In: *arXiv preprint arXiv:2008.01550* (2020).

[139] Liang Zhou, Chris R Johnson, and Daniel Weiskopf. “Data-driven space-filling curves”. In: *IEEE transactions on visualization and computer graphics* 27.2 (2020), pp. 1591–1600.

[140] Yin Zhou and Oncel Tuzel. “Voxelnet: End-to-end learning for point cloud based 3d object detection”. In: *CVPR*. 2018, pp. 4490–4499.

[141] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Wei Li, Yuexin Ma, Hongsheng Li, Ruigang Yang, and Da Lin. “Cylindrical and Asymmetrical 3D

Convolution Networks for LiDAR-based Perception". In: *IEEE transactions on pattern analysis and machine intelligence* PP (2021).

[142] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. "Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation". In: *CVPR* (2021), pp. 9934–9943.

[143] Zhen Zhu, Mengde Xu, Song Bai, Tengteng Huang, and Xiang Bai. "Asymmetric Non-Local Neural Networks for Semantic Segmentation". In: 2019, pp. 593–602.

[144] Jacob Ziv and Abraham Lempel. "Compression of individual sequences via variable-rate coding". In: *IEEE transactions on Information Theory* 24.5 (1978), pp. 530–536.

List of Publications

[1] Wanli Chen, Xufeng Yao, Xinyun Zhang, and Bei Yu. "Efficient Deep Space Filling Curve". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 17525–17534.

[2] Wanli Chen, Xinge Zhu, Guojin Chen, and Bei Yu. "Efficient Point Cloud Analysis Using Hilbert Curve". In: *European Conference on Computer Vision*. Springer. 2022, pp. 730–747.

[3] Wanli Chen, Xinge Zhu, Ruoqi Sun, Junjun He, Ruiyu Li, Xiaoyong Shen, and Bei Yu. "Tensor low-rank reconstruction for semantic segmentation". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVII 16*. Springer. 2020, pp. 52–69.