Efficient Deep Space Filling Curve

Wanli Chen, Xufeng Yao, Xinyun Zhang, Bei Yu
The Chinese University of Hong Kong
{wlchen,xfyao,xyzhang21,byu}@cse.cuhk.edu.hk

Abstract

Space-filling curves (SFCs) act as a linearization approach to map data in higher dimensional space to lower dimensional space, which is used comprehensively in computer vision, such as image/point cloud compression, hashing and etc. Currently, researchers formulate the problem of searching for an optimal SFC to the problem of finding a single Hamiltonian circuit on the image grid graph. Existing methods adopt graph neural networks (GNN) for SFC search. By modeling the pixel grid as a graph, they first adopt GNN to predict the edge weights and then generate a minimum spanning tree (MST) based on the predictions, which is further used to construct the SFC. However, GNN-based methods suffer from high computational costs and memory footprint usage. Besides, MST generation is un-differentiable, which is infeasible to optimize via gradient descent. To remedy these issues, we propose a GNN-based SFC-search framework with a tailored algorithm that largely reduces computational cost of GNN. Additionally, we propose a siamese network learning scheme to optimize DNN-based models in an end-to-end fashion. Extensive experiments show that our proposed method outperforms both DNN-based methods and traditional SFCs, e.g. Hilbert curve, by a large margin on various benchmarks.

1. Introduction

Space-filling curves (SFCs) are performed as the data linearization method which transforms data in n-D to 1D sequence. It has various applications in computer vision tasks, such as image/point cloud compression [22, 40], image transmission [25], data clustering [27], and geography hashing [1]. Currently, SFCs are widely used in deep learning tasks like point cloud analysis [4] and knowledge distillation [**?**]. Traditional SFCs are in fractal structures with many useful properties that can benefit many downstream applications. For example, Hilbert curve [12], with a locality-preserving structure, is widely used in data compression tasks. Z-curve [28] significantly accelerate the hashing problem thanks to its jump-connected structure.

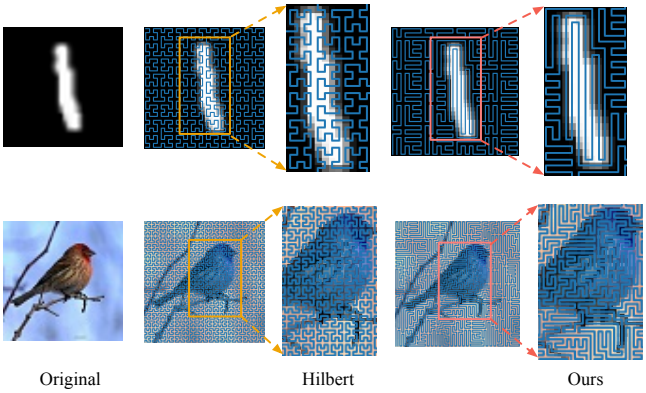


Figure 1: Comparison between our method and Hilbert curve. It can be found that the SFC generated by our model is determined by the image context while Hilbert curve has a fixed structure.

However, with a fixed structure, traditional SFCs lack flexibility when facing different data and tasks, which limits the broader application of SFCs.

To further improve the performance of SFCs, data-adaptive space-filling curves are proposed. (see Figure 1) Previous works [44, 29, 6] mainly focus on the context of image, such as pixel difference and image gradient during SFC generation. Then it can reach better locality-preserving properties compared with Hilbert curve. In these works, the generation of SFCs is transformed into the problem of finding a Hamiltonian path in a grid graph as shown in Figure 2. Currently, following the same SFC modeling scheme, researchers attempt to generate high-quality SFCs via deep learning-based approach [39]. Specifically, they first generate multiple small circuits based on a graph $\mathcal{G}$ and predict the edge weights $\boldsymbol{W}_{\mathcal{G}'}$ in dual graph $\mathcal{G}'$ through graph neural networks (GNN) [17, 38]. Then, they search for a Minimum Spanning Tree (MST) on $\mathcal{G}'$, and finally, a single Hamiltonian circuit is formed according to the MST. Powered by deep neural networks, the learning-based methods outperform traditional data-adaptive SFCs by a large margin. However, the application of such methods is limited by the high computational cost since GNN is very GPU-consuming when dealing with grid graphs ($e.g.$ the adjacency matrix of

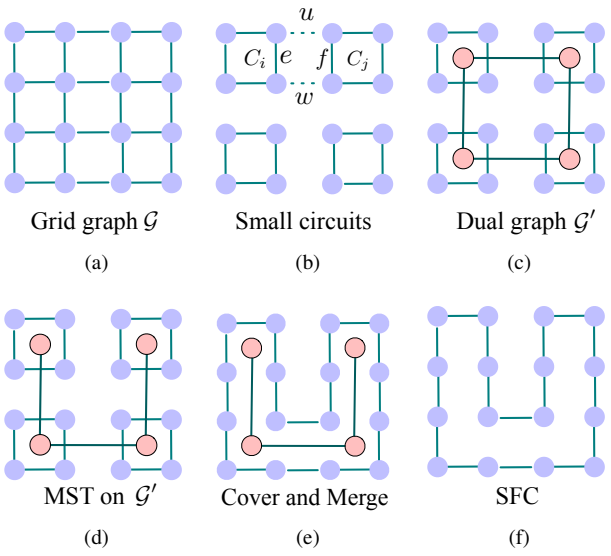


Figure 2: The process of generating data-adaptive SFCs.

a 32×32 grid graph has the size of 1024×1024). Moreover, the MST is generated using greedy algorithm like Prim [31], which is classified as an undifferentiable algorithm. This significantly complicates the optimization of this problem through gradient-based methods.

In this paper, we introduce an efficient deep learning-based SFC generation method to remedy these issues. To tackle the first challenge, we propose an Efficient-GCN (EGCN) module that can significantly reduce computational cost. In addition, we design a novel learning scheme to optimize our model in an end-to-end manner. To be more specific, we first encode input images with convolutional networks [11] and our proposed feature normalization module. Then, we process the obtained features via EGCN. Our EGCN module is based on the fact that the adjacency matrix of a grid graph only contains 4 offset-diagonals. Based on the observation that multiplying an offset-diagonal matrix is identical to shifting the original matrix, our EGCN is free of matrix multiplication of adjacency matrices, which saves computational cost.

Optimization scheme is another important part in our model since finding an optimal SFC is combinatorial optimization problem. However, previous arts [15, 8, 18] are not stable in our task as illustrated in prior work [39]. Therefore, in this paper, we design an elegant solution to find optimal SFCs. Inspired by the self-learning procedure [10, 5], we design a new learning scheme to optimize the given objectives effectively.

On top of that, to increase the stability of our model, we propose a multi-stage MST algorithm that combines the MSTs from multi-stage image features. So that the generated SFCs can receive multi-stage image information and achieve better performance further. Our contribution can be summarized as follow:

- We introduce an Effective-GCN (EGCN) module that tailors grid graphs. It dramatically decreases the memory cost of GCN.
- We propose a new learning scheme to find optimal SFCs in the end-to-end manner.
- We adopt a multi-stage MST algorithm, which helps generate stable SFCs.
- Extensive experiments show that our method outperforms both DNN-based and traditional methods.

2. Related Work

Fractal Space-Filling Curves. Fractal space-filling curves are generated by the infinite iteration of a basic unit $f_1(x)$. In 1890, Peano constructs the first continuous surjective mapping from $\mathbb{R} \to \mathbb{R}^2$, which is known as the Peano space-filling curve [35]. It is formulated as a continuous function $f : [0,1] \to [0,1]^2$ given by $f(x) = \lim_{n\to\infty} f_n(x)$. Cesaro [35] shows the analytic arithmetic expression of Peano curve using parametric equation $t \to (\varphi(t), \psi(t))$ as follow:

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

where

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

A similar fractal SFC is constructed by Lebesgue [28] in 1904 and is known as Z-curve, which is formulated by the $t \to (\varphi(t), \psi(t))$ parametric equation:

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

where $f(t) = 1 - t, t \in [0,1]$. Z-curve has many jump connections that link two distant data points, which makes it qualified for time-sensitive tasks such as geographic hashing [1], data indexing [40] and etc. Hilbert curve $f_h(x)$ is another Fractal SFC with good locality-preserving property [27]. Its analytic expression is described using the quaternary Cantor set as mentioned in [35], in other word, $f_h(x) = f_h\left(0_4 q_1 q_2 q_3 \ldots\right)$ and the Hilbert curve is shown as:

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

where e_{kj} is the number of k's preceding q_j (mod 2) and $d_j = e_{0j} + e_{3j}$ (mod 2). Hilbert curve is welcomed for the

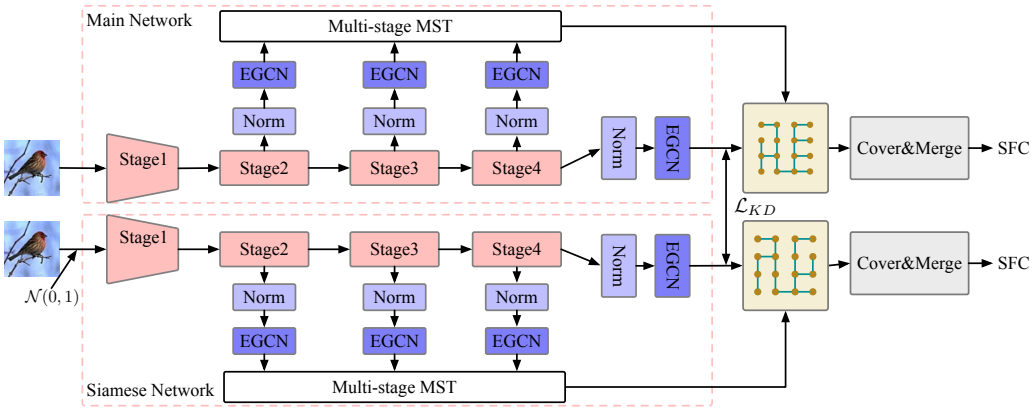


Figure 3: The main framework of our model. Norm represents the feature normalization $\mathrm{L}(\cdot)$ and Cover&Merge is the Cover-and-Merge algorithm. We use ResNet during implementation, which has 4 output stages. The network is optimized by minimizing the KL-divergence between the main network and the siamese network. $\mathcal{N}(0,1)$ represents the standard Gaussian noise. The MSTs are generated via Prim algorithm with $\mathcal{O}(N^2)$ complexity. Since the tested images are relatively small (32× 32 or 64×64) and such complexity is acceptable.

locality-sensitive tasks like image and point cloud compression [40].

Every fractal SFC has its own properties and they are qualified for different tasks. They are cost-friendly since they can be easily generated by repeating basic unit. However, these pre-defined structures cannot perfectly fit the data distribution of various datasets and the curve selection shall be very careful for different tasks. Hence, the data-adaptive space-filling curves are proposed to match the diversity of data and tasks.

Data-adaptive Space-Filling Curves. Compared to fractal SFCs, data-adaptive SFCs are more suitable for data linearization since they are generated according to the data distributions such as image context [6], gradient [29], and neighborhood similarity [44]. These data-adaptive SFCs are formed through the **Cover-and-Merge** algorithm [25, 6, 39].

Specifically, in the **cover** step, we first create an undirected grid graph $\mathcal{G}$ and each node in $\mathcal{G}$ is the location of pixels (see Figure 2(a)). Then we generate several small circuits based on $\mathcal{G}$ (see Figure 2(b)). After that, an undirected dual graph $\mathcal{G}'$ is constructed over $\mathcal{G}$ as shown in Figure 2(c). Finally, the edge weights in $\mathcal{G}'$ are modeled according to different approaches. For example, Dafner *et al.* [6] assign the edge weight $\boldsymbol{W}(C_i, C_j)$ between to nodes C_i and C_j in dual graph as follow:

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

where u,w,e,f are the differences between adjacent pixels (*e.g.* $|u| = |p_1 - p_2|$, see Figure 2(b)). Wang *et al.* [39] further improve the performance of data-adaptive SFC by introducing graph neural networks (GNN) for edge weight prediction as follow:

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

where $\mathfrak{I}$ is the input image and $\boldsymbol{W}$ is the edge weights of $\mathcal{G}'$ and $\mathrm{G}(\cdot)$, $\mathrm{F}(\cdot)$ are GNN and CNN, respectively. Since finding the optimal SFC via $\boldsymbol{W}$ is an NP-hard problem, they design a learning scheme that includes an evaluator network $\mathrm{E}(\cdot)$ for weight optimization.

After determining the edge weights of $\mathcal{G}'$, we move to the **merge** step. Firstly, we find the MST τ in dual graph $\mathcal{G}'$ (see Figure 2(d)) and link all the small circuits in $\mathcal{G}$ according to τ. After that, a new SFC is obtained as shown in Figure 2(f). Cover-and-Merge algorithm transforms SFC generation into the task of finding a Hamiltonian path in $\mathcal{G}$.

3. Methodology

3.1. Problem Formulation

The problem of generating a space-filling curve via deep learning-based approach can be described as follows: Given an input gray image $\mathfrak{I}^{\in 1\times H\times W}$, where H and W are the image height and width, we first split it into many 2×2 grids. Then, for each small grid, all 4 points are connected to form a small circuit, as shown in Figure 2(b). Based on the graph with small circuits, we generate the dual graph [2] $\mathcal{G}'$, as shown in Figure 2(c). Then, we are going to predict the edge weights $\boldsymbol{W}_{\mathcal{G}'}$ in $\mathcal{G}'$. After getting the dual graph weights, we search an MST on $\mathcal{G}'$ according to $\boldsymbol{W}_{\mathcal{G}'}$ (see Figure 2(d)) and apply Cover-and-Merge algorithm [25] to merge the obtained MST with small circuits to get the SFC, as shown in Figure 2(f).

Overview. As shown in Figure 3, our framework consists of two parts: graph weight generation and SFC generation. For graph weight generation, we use a learnable neural network, consisting of a convolutional neural network (CNN) $\mathrm{Conv}(\cdot)$, a feature normalization layer $\mathrm{L}(\cdot)$, and an EfficientGCN $\mathrm{EGCN}(\cdot)$, to extract the graph weights $\boldsymbol{W}_{\mathcal{G}'}$ from the

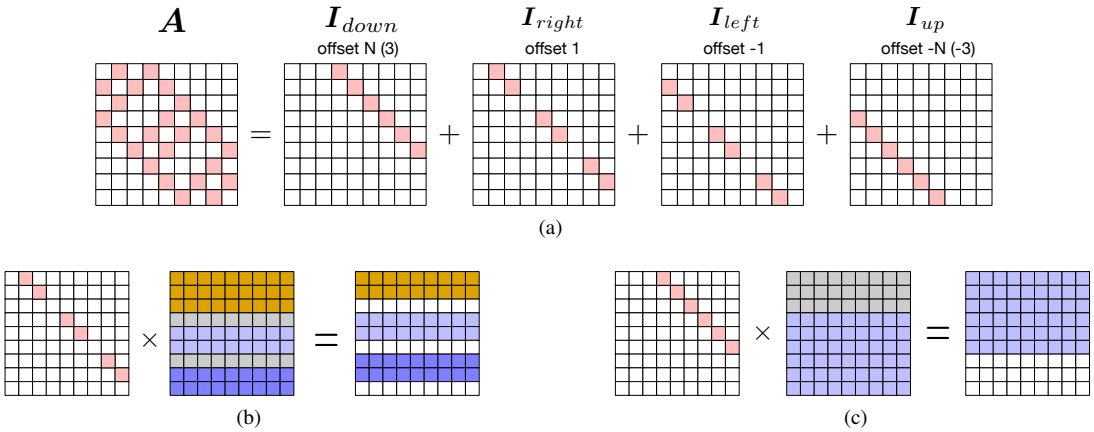


Figure 4: Matrix multiplication with offset diagonal matrices. (a) Adjacency matrix of a 3×3 grid graph; (b) A matrix left multiplied by $\boldsymbol{I}_{right}$ is equivalent to shifting the elements of the matrix upward and replacing some rows (gray rows) with 0; (c) A matrix left multiplied by $\boldsymbol{I}_{down}$ is equivalent to shifting the elements of the matrix upward and padding the last rows with 0. The colored elements (pink, yellow, blue, purple and gray) represents non-zero elements while the uncolored elements stand for 0.

input image, which can be formulated as:

$$\boldsymbol{W}_{\mathcal{G}'} = \mathrm{EGCN}(\mathrm{L}(\mathrm{Conv}(\mathcal{I}))). \tag{1}$$

For SFC generation, we first construct an MST based on the predicted $\boldsymbol{W}_{\mathcal{G}'}$ via Prim algorithm [31] $\mathcal{T}(\cdot)$, and then generate the SFC by Cover-and-Merge algorithm, denoted as:

$$\mathrm{SFC} = \mathrm{Cover\_and\_Merge}(\mathcal{T}(\boldsymbol{W}_{\mathcal{G}'})). \tag{2}$$

Note that the SFC generation procedure is un-differentiable. Therefore, we can not directly adopt gradient descent to optimize the models. To mitigate this issue, we propose a new learning scheme to effectively update the model parameters.

3.2. Graph Weight Generation

Convolutional Network. For every input gray image $\mathcal{I}^{\in 1\times H\times W}$, we first extract its spatial features via residual convolutional networks [11] as follows:

$$\boldsymbol{\mathcal{X}} = \mathrm{F}(\mathcal{I}). \tag{3}$$

Then, we upsample $\boldsymbol{\mathcal{X}}$ to the original image size $H \times W$ via Bilinear interpolation and apply 2D convolution with stride=2 and kernel size=1 to get featuremap $\boldsymbol{W}_I^{C\times \frac{H}{2}\times \frac{W}{2}}$, where C is the output channel number.

Feature Normalization. Next, we model the intermediate graph weights $\boldsymbol{W}_{\mathcal{G}_I}$ by reshaping and normalizing $\boldsymbol{W}_I$. Suppose $N = \frac{H}{2} = \frac{W}{2}$, $\boldsymbol{W}_I$ will be reshaped to $\tilde{\boldsymbol{W}}_I \in \mathbb{R}^{N^2\times C}$. After that, $\boldsymbol{W}_{\mathcal{G}_I}$ is obtained by the Softmax normalization of $\tilde{\boldsymbol{W}}_I$ along C axis ($\mathrm{S}_1(\cdot)$) and N^2 axis ($\mathrm{S}_2(\cdot)$). Then, we concatenate the two features, denoted as:

$$\boldsymbol{W}_{\mathcal{G}_I} = \mathrm{L}(\tilde{\boldsymbol{W}}_I) = \mathrm{Concat}(\mathrm{S}_1(\tilde{\boldsymbol{W}}_I), \mathrm{S}_2(\tilde{\boldsymbol{W}}_I)). \tag{4}$$

Using the intermediate graph weights $\boldsymbol{W}_{\mathcal{G}_I}$ as input, Efficient-GCN is then proposed to determine the edge weights $\boldsymbol{W}_{\mathcal{G}'}$.

Efficient-GCN. Similar to NSFC [39], EGCN aims to capture the long-range dependencies of image context to predict $\boldsymbol{W}_{\mathcal{G}'}$. EGCN is a variant of GCN [17], which has the basic formulation as:

$$\begin{aligned} \boldsymbol{W}_{\mathcal{G}_I}^{r+1} &= \sigma\left(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{W}_{\mathcal{G}_I}^{r}\boldsymbol{W}_k^r\right), \\ \boldsymbol{W}_{\mathcal{G}'} &= \boldsymbol{W}_{\mathcal{G}_I}^{R+1}, \quad r = 1, 2, \ldots, R, \end{aligned} \tag{5}$$

where $\boldsymbol{W}_{\mathcal{G}_I}^r$ is the r_{th} input intermediate graph, $\hat{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A}$, $\boldsymbol{A}$ is the adjacency matrix, $\hat{\boldsymbol{D}}$ represents the degree matrix of $\hat{\boldsymbol{A}}$, $\boldsymbol{W}_k^r$ is the learnable weights for $\boldsymbol{W}_{\mathcal{G}_I}^r$, and $\sigma(\cdot)$ stands for the non-linear operation. NSFC [39] first adopts graph networks of Equation (5) to learn $\boldsymbol{W}_{\mathcal{G}'}$ and generates SFC that has better quality compared with previous works [6, 44, 29]. However, for a gird graph with size $H \times W$, the original GNN requires the modeling of adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{HW\times HW}$, which consumes a lot of computational resources. Luckily, an efficient GNN for grid graphs is achievable after investigating its adjacency matrix $\boldsymbol{A}$.

Specifically, the nodes in a 4-connected grid graph only have edges in up, down, left, and right directions. Therefore, $\boldsymbol{A}$ can be formulated as:

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

where $\boldsymbol{I}_{right} \in \mathbb{R}^{N^2\times N^2}$ is the diagonal matrix with offset 1 and $\boldsymbol{I}_{down} \in \mathbb{R}^{N^2\times N^2}$ is the diagonal matrix with offset N (See Table 7a). $\boldsymbol{I}^\top{}_{down}$ and $\boldsymbol{I}^\top{}_{right}$ represent the transpose

matrix of $\boldsymbol{I}_{down}$ and $\boldsymbol{I}_{right}$, respectively. Then, its multiplication with $\boldsymbol{W}^r_{\mathcal{G}_I} \in \mathbb{R}^{N^2 \times C_r}$ (C_r represents r_{th} input channel number) will have the following formulation:

$$\begin{aligned}\boldsymbol{A}\boldsymbol{W}^r_{\mathcal{G}_I} &= \boldsymbol{I}_{down}\boldsymbol{W}^r_{\mathcal{G}_I} + \boldsymbol{I}_{right}\boldsymbol{W}^r_{\mathcal{G}_I} \\ &+ \boldsymbol{I}^\top{}_{down}\boldsymbol{W}^r_{\mathcal{G}_I} + \boldsymbol{I}^\top{}_{right}\boldsymbol{W}^r_{\mathcal{G}_I} \\ &= \boldsymbol{W}^r_{\mathcal{G}_I N} + \boldsymbol{W}^r_{\mathcal{G}_I 1} + \boldsymbol{W}^r_{\mathcal{G}_I -N} + \boldsymbol{W}^r_{\mathcal{G}_I -1},\end{aligned} \tag{6}$$

where $\boldsymbol{W}^r_{\mathcal{G}_I N}$ represents that all elements $e \in \boldsymbol{W}^r_{\mathcal{G}_I}$ moving upwards by N and padding the last N rows with 0 (see Figure 4(c)). Similarly, $\boldsymbol{W}^r_{\mathcal{G}_I -N}$ means that $e \in \boldsymbol{W}^r_{\mathcal{G}_I}$ moving downward by N and padding the first N rows with 0. The same process will be applied to $\boldsymbol{W}^r_{\mathcal{G}_I 1}$ and $\boldsymbol{W}^r_{\mathcal{G}_I -1}$. The only difference is that $\boldsymbol{W}^r_{\mathcal{G}_I 1}$ and $\boldsymbol{W}^r_{\mathcal{G}_I -1}$ should remove some rows after element shifting (see Table 7b for details.)

Therefore, our proposed EGCN is achieved by the element shifting of $\boldsymbol{W}^r_{\mathcal{G}_I}$. This operation has a smaller computational cost since we do not perform **real** matrix multiplication between adjacency matrix $\boldsymbol{A}$ and input features. Additionally, it does not cost GPU memory for generating $\boldsymbol{A}$. Using the computation scheme of Equation (6), given an input intermediate graph with weight $\boldsymbol{W}_{\mathcal{G}_I}$, the proposed EGCN is formulated as:

$$\begin{aligned}\boldsymbol{W}^{r+1}_{\mathcal{G}_I} &= \mathrm{EGCN}(\boldsymbol{W}^r_{\mathcal{G}_I}, \boldsymbol{A}) = \sigma\left((\boldsymbol{I} + \boldsymbol{A})\boldsymbol{W}^r_{\mathcal{G}_I}\boldsymbol{W}^r_k\right) \\ &= \sigma((\boldsymbol{W}^r_{\mathcal{G}_I} + \boldsymbol{W}^r_{\mathcal{G}_I N} + \boldsymbol{W}^r_{\mathcal{G}_I 1} + \boldsymbol{W}^r_{\mathcal{G}_I -N} \\ &\quad + \boldsymbol{W}^r_{\mathcal{G}_I -1})\boldsymbol{W}^r_k), \quad r = 1, 2, \ldots, R-1.\end{aligned} \tag{7}$$

In our model, the learnable weight $\boldsymbol{W}^r_k$ is implemented by spectral graph convolution. For the last GNN layer (R_{th} layer), we follow Equation (7) but set the output channel number to 1 and then remove the non-linear function $\sigma(\cdot)$. The output of EGCN $\boldsymbol{W}^{R+1}_{\mathcal{G}_I} \in \mathbb{R}^{N^2}$ represents the node weights of graph $\mathcal{G}'$ and the edge weight, $\boldsymbol{W}_{\mathcal{G}'}$, between two nodes will be calculated by the average value of their node weights. Then, we construct an MST based on $\boldsymbol{W}_{\mathcal{G}'}$. However, during experiments, we find the obtained MST $\mathcal{T}(\boldsymbol{W}_{\mathcal{G}'})$ is not stable, which means the predicted SFCs vary severely with different random seeds during training and it makes the resultant SFC not constantly have good performance. Therefore, we propose a multi-stage Minimum Spanning Tree generation algorithm which is based on multi-level image features to mitigate the stability problem.

3.3. Multi-Stage Minimum Spanning Tree

Considering a CNN [43, 19, 36, 11], the shallow layers capture more low-level information, such as texture and edges, while the deep layers contain rich high-level semantic information. Merging multi-level image features will increase the stability of outputs and this approach is proved to be successful in various tasks [23, 24, 33, 26]. Inspired by these previous works, we build a multi-stage minimum spanning tree generation algorithm for stable $\boldsymbol{W}_{\mathcal{G}'}$ prediction.

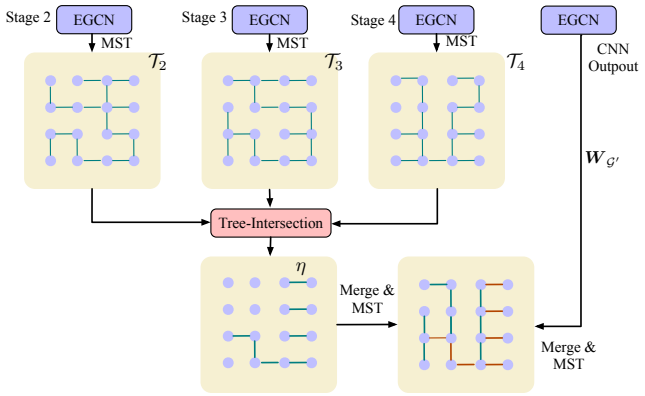


Figure 5: Illustration of multi-stage MST. After tree-intersection, the obtained graph edge weights are set to η, which is designed to make sure that these edges will be included in the next MST.

Specifically, we take the second, the third and the fourth stage output of ResNet as:

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

where $\mathfrak{X}_m \in \mathbb{R}^{C_m \times H_m \times W_m}$ and $m \in \{2, 3, 4\}$ is the output of m_{th} stage of ResNet. Then we generate corresponding MST τ_2, τ_3 and τ_4 based on different levels of feature:

$$\tau_m = \mathcal{T}(\mathrm{EGCN}(\mathrm{L}(\mathfrak{X}_m))), \quad m \in \{2, 3, 4\}. \tag{8}$$

After that, we define a **tree-intersection** operation for spanning trees τ_1 and $\tau_2 \cdots \tau_n$:

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

where $\boldsymbol{A}_{\tau_n}$ is the adjacency matrix of τ_n and $\cdot$ represents element-wise production. In the real implementation, we only store the 4 diagonals of $\boldsymbol{A}_{\tau_n}$ to save GPU memory. The multi-stage MST generation algorithm for edge weights $\boldsymbol{W}_{\mathcal{G}'}$ is performed as follows:

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

where τ_2, τ_3 and τ_4 are generated by Equation (8) and η is a relatively small number ($e.g.$ $1e-3$). $\boldsymbol{A}_{\boldsymbol{W}_{\mathcal{G}'}}$ represents the adjacency matrix of $\boldsymbol{W}_{\mathcal{G}'}$ and $\boldsymbol{A}^M_{\boldsymbol{W}_{\mathcal{G}'}}$ is its multi-stage adjacency matrix. If $\boldsymbol{W}_{\mathcal{G}'}$ is generated through $\boldsymbol{A}^M_{\boldsymbol{W}_{\mathcal{G}'}}$, $\mathcal{T}(\boldsymbol{W}_{\mathcal{G}'})$ will have relatively small cost when passing through the edges that existing in τ_2, τ_3 and τ_4. In this way, we can filter out some edges that are impossible to form MST and increase the probability of generating a new MST by selecting the edges that exist in all previous MSTs (see Figure 5).

4. Weight Optimization

SFC generation is un-differentiable because of MST generation, $e.g.$, Prim algorithm [31], and Cover-and-Merge algorithm. It can not be directly optimized by gradient-based

methods. Therefore, we address this problem by designing a new learning scheme, which is detailed as follows.

4.1. Objectives

Note that the generated SFCs should be adapted for different tasks. Therefore, in this paper, we investigate SFCs in two task: data transmission and data compression and they can be measured by autocorrelation and LZW code length respectively. Therefore, we use them as the objectives.

Autocorrelation. Autocorrelation factor is widely used in various tasks like data transmission, spectral analysis [25, 37, 14]. Given a flattened gray image sequence $\boldsymbol{Y} \in \mathbb{R}^{HW}$, autocorrelation evaluates the inherent similarity of the sequence by calculating the correlation factor between $\boldsymbol{Y}[t], t \in \{1, \cdots, HW\}$ with its delay copy $\boldsymbol{Y}[t+k]$:

$$\Phi_a = \frac{\sum_{i=1}^{HW-k} \boldsymbol{Y}[t]\boldsymbol{Y}[t+k]}{\sum_{i=1}^{HW} \boldsymbol{Y}[t]^2}, \tag{9}$$

where $k \in \{1, \cdots, HW\}$ is the delay factor. The larger autocorrelation, the better the quality of obtained SFC.

LZW Code Length. LZW Coding [45] is a lossless data compression algorithm. Given a flattened pixel sequence $\boldsymbol{Y}$, the quality of LZW coding can be measured using the length of the LZW encoded sequence as follow:

$$\Phi_l = \text{Length(LZW Coding}(\boldsymbol{Y})). \tag{10}$$

The smaller LZW Code length, the better compression efficiency of obtained SFCs.

4.2. Learning Scheme

Inspired by previous generative models [9, 32], self-learning [5, 10] and knowledge distillation schemes [13, 34], we design two networks: main network and siamese network, as shown in Figure 3. The input of main network is the original image $\mathcal{I}$ while the input of the siamese network is the original image with some noise ($\mathcal{I} + \mathcal{N}(0, 1)$). We add random noise to the siamese network because random noise only changes the intensity of the pixels while keeping important features like edges intact. Φ^s, Φ^m indicate the evaluation score produced by the siamese network and main network under the given objectives. We only optimize one network per iteration. If Φ^s is better, the siamese network will be the teacher model and main network will be the student. Otherwise, the main network will be the teacher. The parameters of student model are updated by minimizing the KL-divergence between the main network and the siamese network, denoted as:

$$\mathcal{L}_{KD} = \begin{cases} \text{KL}\left(\boldsymbol{e}^s \| \boldsymbol{e}^m\right), & \text{If } \Phi^m \text{ is better,} \\ \text{KL}\left(\boldsymbol{e}^m \| \boldsymbol{e}^s\right), & \text{If } \Phi^s \text{ is better,} \end{cases} \tag{11}$$

where $\boldsymbol{e}^s$, $\boldsymbol{e}^m$ represent the outputs of siamese network and main network, respectively. In the testing stage, we only use main network for inference. Our learning scheme is more efficient than NSFC and we can generate SFC for each image instead of each class like NSFC. More details about network training can be found in **Appendix**.

5. Experiments

Datasets. We use gray image datasets MNIST [21] and Fashion-MNIST [42] during experiments. Both datasets have 60000 training samples and 10000 testing samples. Following the setting in NSFC [39], we change the original image to 32×32 before training and testing. Additionally, Tiny-imagenet [20] experiments are involved in the following experiments. It is sampled from ImageNet dataset [7] and the size of all images are 64×64. We resize all input images to 32×32 for quantitative comparison. Additionally, 64×64 images are used to show the scalablity of our model. It has 200 object classes and each class contains 500/50/50 training/validation/test images. We will use training and validation set during experiments.

Experimental Setting. Our method is implemented by Pytorch framework [30] and we apply ResNet-18 during experiments. All images will be transformed into grayscale before input. The model is trained from scratch and the number of GNN layer R (refered in Equation (7)) is set to 3 for all experiments. We set the training batch size to 128 and the total epoch is set to 80. In the training stage, we decay learning rate by half in every 20 epochs. We use Adam [16] optimizer with learning rate $lr = 0.001$ for all experiments.

5.1. Quantitative Comparison

Different Objectives. We train our model using different objectives including autocorrelation and LZW code length. The experimental results can be found in Table 1.

During experiments, we first train our model and NSFC using autocorrelation with delay factor $k = 6$. After that, we generate SFCs via Zigzag, Hilbert, and Dafner [6] scheme sequentially. Then we test the average autocorrelation of all obtained SFCs. After autocorrelation experiments, we optimize our proposed framework and NSFC according to the LZW code length. Table 1 illustrates that our model significantly outperforms other methods in all evaluation metrics. For example, under the LZW coding scheme, our method only occupies an average of 158.3 bytes in the MNIST dataset, which is much shorter than previous works. Some visualized results on Tiny-Imagenet dataset can be found on Figure 6.

Computational Cost. In this part, we compare the computational cost of our proposed EGCN with other methods

Dataset	Method	Autocorrelation ↑	LZW Code Length (bytes)↓
MNIST [21]	Zigzag	0.207	175.4
	Hilbert [12]	0.475	182.7 (+7.3)
	Dafner [6]	0.401	-
	NSFC [39]	0.558	171.1 (-4.3)
	Ours	**0.625**	**158.3 (-17.1)**
Fashion-MNIST [42]	Zigzag	0.552	425.8
	Hilbert [12]	0.723	427.3(+1.5)
	Dafner [6]	0.704	-
	NSFC [39]	0.786	412.4 (-13.4)
	Ours	**0.834**	**400.7 (-25.1)**
Tiny-imagenet (32×32) [20]	Zigzag	0.811	925.1
	Hilbert [12]	0.874	927.6 (+2.5)
	Dafner [6]	0.896	909.0 (-16.1)
	NSFC [39]	0.913	904.9 (-20.2)
	Ours	**0.936**	**888.7 (-36.4)**
Tiny-imagenet (64×64) [20]	Zigzag	0.719	-
	Hilbert [12]	0.773	-
	Dafner [6]	0.779	-
	NSFC [39]	-	-
	Ours	**0.826**	-

Table 1: Comparison between different methods. Zigzag curve is implemented by Pytorch reshape function. Additional to the 32×32 images, we use Tiny-imagenet with size 64×64 for scalability experiments. The results show that our model is scalable to a larger image.

Method	GPU	Params	Inference time
GCN [17]	104M	1.6M	11ms
GAT [38]	120M	1.8M	12ms
SGC [41]	88M	0.8M	9ms
FastGCN [3]	96M	0.5M	8ms
EGCN	**32M**	**0.3M**	**4ms**

Table 2: Comparison between different GCN methods. All methods are tested by using grid graphs with size 64×64 and batch size 512.

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

Table 3: Ablation study on multi-stage MST.

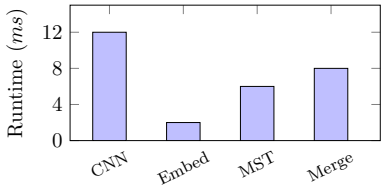


Table 4: Average time consumption of our model in inference stage.

Learning Scheme	AC	Training Time/epoch	Params
NSFC [39]	0.593	1867s	22.9M
Ours	**0.625**	**72s**	22.3M

Table 5: Comparison between our scheme and NSFC.

in terms of GPU occupation and inference time. In the experiments of Table 2, we input a batch of equal-weight grid graphs with size 64×64, and the layer number for all methods is set to 3 for a fair comparison. Table 2 shows that our proposed EGCN not only significantly outperforms the classical GCN [17, 38] in terms of GPU memory occupation and inference time, but also surpasses other fast GCN methods such as FastGCN [3] and SGC [41].

To demonstrate the efficiency of our proposed EGCN, we present the average time consumption of each component of our model in MNIST experiments, as depicted in Figure 4. The terms "Embed" and "Merge" refer to the graph weight embedding and Cover-and-Merge algorithm, respectively. Specifically, the CNN, MST, and Cover-and-Merge algorithm require 12ms, 5ms, and 8ms, respectively. In contrast, our proposed graph weight embedding scheme only requires 2ms, highlighting the effectiveness of EGCN.

5.2. Ablation Study

In this part, we ablate all modules in our framework. MNIST is used for the following experiments and the training/testing objective is autocorrelation with delay factor $k = 6$.

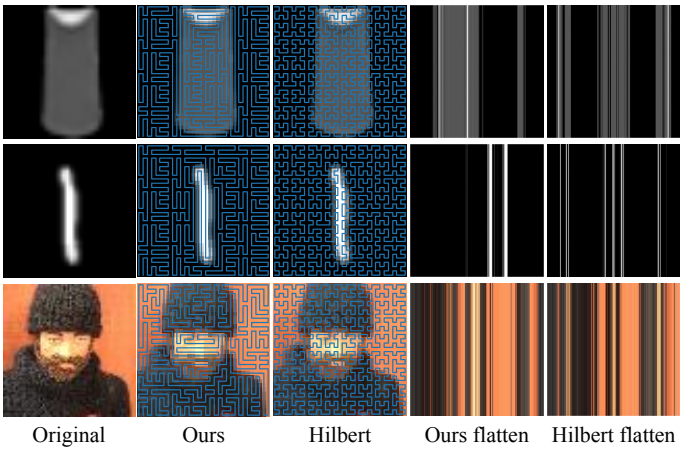
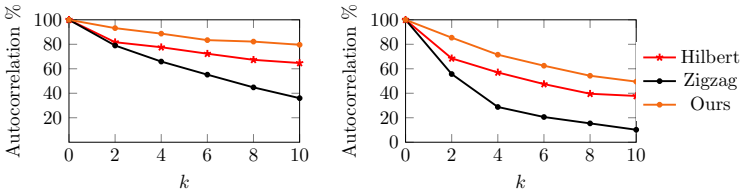


Figure 6: Visualized results of our proposed methods. We first show SFCs generated by our method. Then we flatten images according to the curve, obtaining an 1D sequence with size HW. Finally, we upsample the sequences to images. It can be found that the images flattened by our method have a higher degree of aggregation, which demonstrates that our SFCs have better clustering property.

Multi-Stage Minimum Spanning Tree. We first validate

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

Table 6: Ablation study on EGCN and feature normalization.



(a) MNIST (b) Fashion-MNIST

Figure 7: Visualization of different autocorrelation factors.

the importance of the proposed Multi-Stage MST, which is introduced to increase network stability in the training process. We use the multi-stage MST algorithm on the output of different stages of ResNet, the results can be found in Table 3. It tells that multi-stage MST greatly reduces the volatility of the predicted results (0.577 ($\pm$0.04) to 0.625($\pm$0.01)). Without multi-stage MST, the generated SFCs do not constantly have good performance. Also, we found that combining the outputs of all 3 stages reaches the best performance. As it is shown in Figure 5, multi-stage MST selects the intersection of all MSTs, which merges multi-level image features. Then, our networks can produce more stable outputs benefiting from rich context features.

Different Learning Schemes. Our learning scheme differs significantly from NSFC [39], which adopts a generator-evaluator structure. In the subsequent experiments, we compare the performance of the two learning schemes based on autocorrelation performance and network training time of one epoch. To conduct these experiments, we create an SFC evaluator according to the settings described in NSFC [39], while keeping other components unchanged (e.g., multi-stage MST, EGCN). The performance of the two learning schemes is shown in Table 5. The results indicate that our learning scheme not only performs better but also has a shorter training time, as we avoid using the time-consuming Dafner [6] algorithm as input.

EGCN and Feature Normalization. This section evaluates the effectiveness of the proposed EGCN and feature normalization layer. Initially, we replace EGCN with the original GCN layer, while maintaining other settings such as learning scheme and multi-stage MST. Subsequently, we substitute GCN with EGCN. According to the results presented in Table 6, the EGCN yields a slight improvement in autocorrelation performance, but the gain is not significant. However, with the aid of feature normalization, our proposed method surpasses GCN by a large margin. We also combine GCN with our feature normalization layer, but the obtained results indicate that the autocorrelation improvement is less pronounced than that of EGCN.

Scalability. Scalability is the major concern of learning-based SFC generation methods because firstly GCN is not suitable for large images since the node number increases exponentially with image size. Secondly, it is still a very challenging problem to quickly find MST on large images. However, due to the efficiency of our proposed EGCN, the problem of the computational cost of GCN is alleviated. To show the scalability of our proposed method, we conduct 64×64 experiments on the Tiny-imagenet dataset using autocorrelation (k=100) objective. The results in Table 1 tell that our model is scalable to the larger image. Additionally, applying the scale-up method mentioned in NSFC [39] is also a good solution to the scalability problem. More experiments about the scalability of our method can be found in **Appendix**.

More Visualization. We conduct experiments using different autocorrelation factors k. Here we visualize the autocorrelation performance with different k on Figure 7. It shows that our method consistently outperforms Hilbert curve and Zigzag curve.

6. Conclusion and Discussions

In this paper, we propose an efficient SFC generation method via deep learning. To be specific, we design a GCN module EGCN to accelerate GCN computation in grid graphs. Additionally, we develop a siamese network-based learning scheme to optimize our network efficiently. Moreover, we propose a multi-stage MST module to stabilize the training process. The experimental results show that our method significantly outperforms previous works such as Hilbert, NSFC by a large margin.

Limitations and Future Works. Searching for an optimal SFC is an NP-hard problem, and solving the problem in a large graph is still challenging. 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 [4], data clustering [27] and *etc.*

References

[1] 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*, pages 290–298. Springer, 2012. 1, 2

[2] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976. 3

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018. 7

[4] Wanli Chen, Xinge Zhu, Guojin Chen, and Bei Yu. Efficient point cloud analysis using hilbert curve. 1, 8

[5] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021. 2, 6

[6] Revital Dafner, Daniel Cohen-Or, and Yossi Matias. Context-based space filling curves. In *Computer Graphics Forum*, volume 19, pages 209–218. Wiley Online Library, 2000. 1, 3, 4, 6, 7, 8

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 6

[8] 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*, pages 170–181. Springer, 2018. 2

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 6

[10] 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. *Advances in neural information processing systems*, 33:21271–21284, 2020. 2, 6

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 2, 4, 5

[12] David Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. In *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes*, pages 1–2. Springer, 1935. 1, 7

[13] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 6

[14] Bogdan Kasztenny. A new method for fast frequency measurement for protection applications. In *13th International Conference on Developments in Power System Protection*, 2016. 6

[15] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017. 2

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 4, 7

[18] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018. 2

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012. 5

[20] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015. 6, 7

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6, 7

[22] Jan-Yie Liang, Chih-Sheng Chen, Chua-Huang Huang, and Li Liu. Lossless compression of medical images using hilbert space-filling curves. *Computerized Medical Imaging and Graphics*, 32(3):174–182, 2008. 1

[23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, pages 2117–2125, 2017. 5

[24] 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*, pages 3431–3440, 2015. 5

[25] Yossi Matias and Adi Shamir. A video scrambling technique based on space filling curves. In *Conference on the theory and application of cryptographic techniques*, pages 398–417. Springer, 1987. 1, 3, 6

[26] 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)*, pages 565–571. IEEE, 2016. 5

[27] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on knowledge and data engineering*, 13(1):124–141, 2001. 1, 2, 8

[28] 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*, pages 326–336, 1986. 1, 2

[29] 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)*, pages 437–442. IEEE, 2011. 1, 3, 4

[30] 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. *Advances in neural information processing systems*, 32, 2019. 6

[31] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. 2, 4, 5

[32] 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. *Advances in neural information processing systems*, 29, 2016. 6

[33] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 5

[34] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 6

[35] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012. 2

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 5

[37] Jan Van Sickle. *GPS for land surveyors*. CRC press, 2008. 6

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 1, 7

[39] Hanyu Wang, Kamal Gupta, Larry Davis, and Abhinav Shrivastava. Neural space-filling curves. *arXiv preprint arXiv:2204.08453*, 2022. 1, 2, 3, 4, 6, 7, 8

[40] Jun Wang and Jie Shan. Space filling curve based point clouds index. In *Proceedings of the 8th International Conference on GeoComputation*, pages 551–562. Citeseer, 2005. 1, 2, 3

[41] F. Wu, T. Zhang, Ahd Souza, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. 2019. 7

[42] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 6, 7

[43] 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*, pages 2921–2929, 2016. 5

[44] Liang Zhou, Chris R Johnson, and Daniel Weiskopf. Data-driven space-filling curves. *IEEE transactions on visualization and computer graphics*, 27(2):1591–1600, 2020. 1, 3, 4

[45] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978. 6