Revisiting CNNs for Trajectory Similarity Learning

Zhihao Chang
School of Software Technology,
Zhejiang University, China
changzhihao@zju.edu.cn

Linzhu Yu
Zhejiang University, China
linzhu@zju.edu.cn

Huan Li
Hangzhou High-Tech Zone (Binjiang)
Institute of Blockchain and Data
Security
Zhejiang University, China
lihuan.cs@zju.edu.cn

Sai Wu
Zhejiang University, China
wusai@zju.edu.cn

Gang Chen
Zhejiang University, China
cg@zju.edu.cn

Dongxiang Zhang*
Hangzhou High-Tech Zone (Binjiang)
Institute of Blockchain and Data
Security
Zhejiang University, China
zhangdongxiang@zju.edu.cn

ABSTRACT

Similarity search is a fundamental but expensive operator in querying trajectory data, due to its quadratic complexity of distance computation. To mitigate the computational burden for long trajectories, neural networks have been widely employed for similarity learning and each trajectory is encoded as a high-dimensional vector for similarity search with linear complexity. Given the sequential nature of trajectory data, previous efforts have been primarily devoted to the utilization of RNNs or Transformers.

In this paper, we argue that the common practice of treating trajectory as sequential data results in excessive attention to capturing long-term global dependency between two sequences. Instead, our investigation reveals the pivotal role of local similarity, prompting a revisit of simple CNNs for trajectory similarity learning. We introduce ConvTraj, incorporating both 1D and 2D convolutions to capture sequential and geo-distribution features of trajectories, respectively. In addition, we conduct a series of theoretical analyses to justify the effectiveness of ConvTraj. Experimental results on four real-world large-scale datasets demonstrate that ConvTraj achieves state-of-the-art accuracy in trajectory similarity search. Owing to the simple network structure of ConvTraj, the training and inference speed on the Porto dataset with 1.6 million trajectories are increased by at least 240x and 2.16x, respectively.

PVLDB Reference Format:
Zhihao Chang, Linzhu Yu, Huan Li, Sai Wu, Gang Chen, and Dongxiang Zhang. Revisiting CNNs for Trajectory Similarity Learning. PVLDB, 18(4): 1013 - 1021, 2024.
doi:10.14778/3717755.3717762

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Proudc/ConvTraj.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 4 ISSN 2150-8097.
doi:10.14778/3717755.3717762

1 INTRODUCTION

Trajectory similarity plays a fundamental role in numerous trajectory analysis tasks. Numerous distance measures, such as Discrete Frechet Distance (DFD) [3], the Hausdorff distance [4], Dynamic Time Warping (DTW) [35], and Edit Distance on Real sequence (EDR) [13], have been proposed and employed in a wide spectrum of applications, including but not limited to trajectory clustering [1, 6], anomaly detection [20, 36], and similar retrieval [25, 29, 37].

Generally speaking, these distance measures involve the optimal point-wise alignment between two trajectories. The distance calculation often relies on dynamic programming and incurs quadratic computational complexity. This limitation poses a significant constraint, particularly when confronted with large-scale datasets with long trajectories. In recent years, trajectory similarity learning has emerged as the mainstream approach to mitigate the computational burden. The main idea is to encode each trajectory sequence T_i into a high-dimensional vector V_i such that the real distance between T_1 and T_2 can be approximated by the distances between their derived vectors V_1 and V_2. Consequently, the complexity of distance calculation can be reduced from quadratic to linear.

Given the sequential nature of trajectory data, existing methods for trajectory similarity learning can be categorized into RNN-based or Transformer-based. RNN-based methods, including NeuTraj [32], Traj2SimVec [38], and T3S [31], employ RNN or its variants (e.g, GRU [14], LSTM [19]) as the core encoder, which can be augmented with additional components such as spatial attention memory in NeuTraj and point or structure matching mechanisms in Traj2SimVec and T3S to enhance performance. Due to the success of Transformer in NLP, TrajGAT [33] and TrajCL [7] adopt Transformer to learn trajectory embedding, which can effectively capture the long-term dependency of sequences.

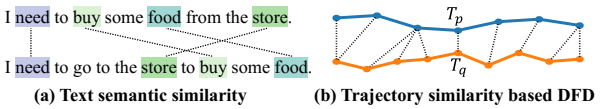
However, we argue that these common practices pay excessive attention to capturing long-term global dependency between two trajectories while ignoring point-wise similarity, which may potentially yield adverse effects. Instead, we should pay more attention to point-wise similarity in the local context. In support of this argument, we conducted an experiment on Porto[1] dataset to evaluate

[1]https://www.kaggle.com/competitions/pkdd-15-predict-taxi-service-trajectory-i/data

Table 1: Performance of Transformer with different attention window sizes. We report the hit rates for two measures: DFD and DTW. The dataset includes 6000 items selected from Porto, with 3000 for training, 1000 for query, and 2000 as the candidate set.

Method	# Paras	(Train time Per Epoch) * # Epochs	Inference time	DFD				DTW			
				HR@1	HR@5	HR@10	HR@50	HR@1	HR@5	HR@10	HR@50
global attention	3.38M	17.28s * 1000	3.58s	22.10	32.58	39.11	50.11	29.40	46.22	54.60	63.41
local attention ($w = 10$)	3.38M	17.28s * 1000	3.58s	**23.20**	**36.74**	**42.80**	**54.70**	31.50	**48.60**	**54.92**	**65.06**
local attention ($w = 5$)	3.38M	17.28s * 1000	3.58s	21.80	35.40	41.83	54.42	**33.60**	46.72	52.34	63.03
1D CNN	0.17M	1.03s * 200	0.16s	**33.23**	**43.94**	**50.84**	**64.78**	30.90	46.66	53.36	**65.14**

the effect of applying Transformer for trajectory encoding with different sizes of attention windows. The first variant is the original Transformer with global attention, where each token engages in self-attention by querying all other tokens. We also implemented two alternative variants with local attention, in which each token only queries its neighbors within a window of count w, i.e., the attention weights outside the window have been masked. We can



(a) Text semantic similarity **(b) Trajectory similarity based DFD**

Figure 1: Texts feature intercrossed matching pairs, whereas trajectories do not.

observe from Table 1 that local attention has great potential to significantly outperform global attention. We explain that existing trajectory distance measurements are alignment-based and the edges for matching pairs are not intercrossed (as shown in Figure 1). This property differs significantly from handling text data in NLP.

These observations reveal the pivotal role of *local similarity*. Instead of adopting Transformer with masked local attention, we are interested in revisiting CNNs in the task of trajectory similarity learning. The reason is that CNNs can also well capture local similarity while offering the advantages of simplicity. As shown in Table 1, with only 5% of the parameters, a simple 1D CNN can remarkably outperform vanilla Transformers after convergence on the DFD. Although slightly lower than local attention on the DTW, 1D CNN has great advantages in efficiency. To further exploit the potential of CNNs, we present ConvTraj with two types of convolutions. We first use 1D convolution to capture the sequential features of trajectories. Then we represent the trajectory as a single-channel binary image and use 2D convolution to capture its geo-distribution. Finally, these features are fused as complementary clues to capture trajectory similarity. To justify the effectiveness of ConvTraj, we conduct a series of theoretical analyses. We prove that 1D convolution and 1D max-pooling can preserve effective distance bounds after embedding, and trajectories located in distant areas yield large distances via 2D convolution, all of which play an important role in trajectory similarity recognition.

We conducted extensive experiments to evaluate the performance of ConvTraj on four real-world datasets. Experimental results show that ConvTraj achieves state-of-the-art accuracy for similarity retrieval on four commonly used similarity measurements, including DFD, DTW, Hausdorff, and EDR. Furthermore, ConvTraj is at least 240x faster in training speed and 2.16x faster in inference speed, when compared with methods based on RNN and Transformer on the Porto dataset containing 1.6 million trajectories.

Our contributions are summarized in the following:

- We argue that trajectory similarity learning should pay more attention to local similarity.
- We present a simple and effective ConvTraj with two types of CNNs for trajectory similarity computation.
- We conduct some theoretical analysis to help justify why such a simple ConvTraj can perform well.
- Extensive experiments on four real-world large-scale datasets established the superiority of ConvTraj over state-of-the-art works in terms of accuracy and efficiency.

2 RELATED WORK

2.1 Heuristic Trajectory Similarity Measures

Heuristic measures between trajectories are derived from the distance between matching point pairs, these measures fall into three categories: (1) *Linear-based* methods [2, 8] only need scan trajectories once to calculate their similarity but may lead to sub-optimal point matches. (2) *Dynamic programming-based* methods are proposed to tackle this issue, such as DTW [35], DFD [3], and others [12, 13, 24, 27]. However, these measurements involve the optimal point-wise alignment between two trajectories without intercrossing between matching pairs and often incur quadratic complexity. Thus it poses significant challenges for similarity search from a large-scale dataset with long trajectories. (3) *Enumeration-based* methods calculate all point-to-trajectory distance, i.e., the minimum distance between a point to any point on a trajectory, then aggregate it. For example, OWD [22] uses the average point-to-trajectory distance, while Hausdorff [4] uses the maximum.

2.2 Learning-based Trajectory Similarity

In recent years, the field of trajectory similarity has witnessed a paradigm shift, primarily fueled by the progress in deep representation learning. This advancement has led to the development of numerous methodologies aimed at encoding trajectories into embedding spaces. Broadly, these approaches can be classified into three categories: (1) *Learn a model to approximate a measurement.* The purpose of these methods is to learn a neural network so that the distance in the embedding space can approximate the true distance between trajectories. Early attempts were generally based on recurrent neural networks, including NeuTraj [32], Traj2SimVec [38], and T3S [31]. Subsequently, some studies tried to capture the long dependency of trajectories based on Transformer [7, 33]. (2) *No given measurements are required to generate training signals.* These

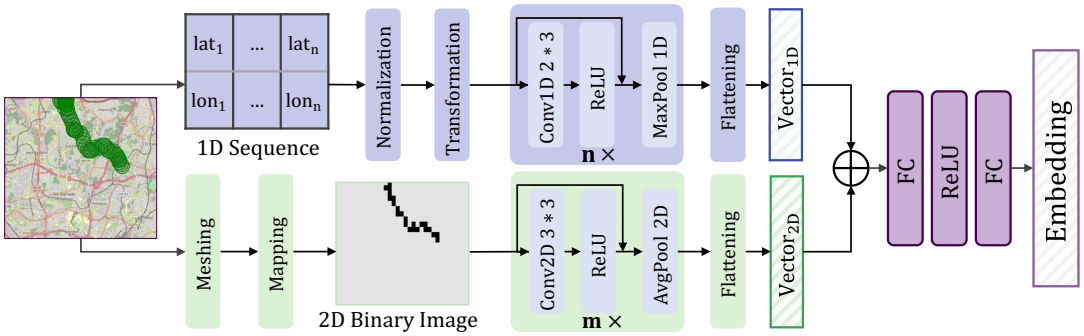


Figure 2: Input preprocessing and network structure of ConvTraj.

methods encode trajectories without the need to generate supervised signals based on measurements. Its purpose is to overcome the limitations of traditional measures such as non-uniform sampling rates and noise. These methods can be divided into RNN-based, including traj2vec [34], t2vec [21], E2DTC [15], etc., CNN-based TrjSR [5], and Transformer-based TrajCL [7]. (3) *Road networks-based.* There have been some studies on trajectory similarity based on road networks [9, 16, 17, 40, 41]. These works use graph neural networks to encode road segments. Since such works introduce relevant knowledge from road networks, we consider them as different research directions and will not delve into these methods.

TrjSR [5] is a well-known CNN-based method for trajectory similarity. It maps trajectories into 2D images and uses super-resolution techniques. However, TrjSR loses the sequential features of trajectories, making it unable to differentiate between two trajectories with the same path but opposite directions. Our ConvTraj uses both 1D and 2D convolutions as the backbone and achieves better results.

3 PROBLEM DEFINITION

Definition 1 (Trajectory)**.** A trajectory T is a series of GPS points ordered by timestamp t, and each point p is a location containing latitude and longitude. Formally, a trajectory $T \in \mathbb{R}^{l\times 2}$ can be denoted as $T = [p_1, ..., p_l]$, with $p_i = (p_i^{lat}, p_i^{lon})$ is the i-th location.

Definition 2 (Trajectory Measure Embedding)**.** Given a specific trajectory similarity measure $f(\cdot,\cdot)$, trajectory measure embedding aims to learn an approximate projection function g, such that for any pair of trajectories T_i with T_j, the distance in the embedding space approximates the true distance between T_i and T_j, i.e., $f(T_i, T_j) \approx d(g(T_i), g(T_j))$. Besides, the vectors in the embedding space should maintain the distance order of true distance, i.e., for any three trajectories T_i, T_j, and T_k, with $f(T_i, T_j) < f(T_i, T_k)$, we should ensure that $d(g(T_i), g(T_j)) < d(g(T_i), g(T_k))$. Here, $f(\cdot,\cdot)$ can be DFD, DTW, or any other measurements. At the same time, $d(\cdot,\cdot)$ is a measure between high-dimensional embedding vectors in the embedding space, such as Euclidean distance, Cosine distance, etc.

4 METHODOLOGY

4.1 Input Preprocessing

Suppose there is a trajectory T containing l GPS points. To process T as the input of our ConvTraj, we perform the following two steps covering both one-dimensional and two-dimensional.

One-dimensional Input. The input of our 1D convolution is a sequence, we thus treat the trajectory T as a sequence with length l and width 2 (i.e., lat and lon). For each point of T, we first normalize it using a min-max normalization, and then apply a multi-layer perceptron (MLP) to perform a nonlinear transformation for each point, thus the trajectory can be processed as a sequence Seq_{1D}.

Two-dimensional Input. The input of our 2D convolution is a binary image, we thus perform the following substeps to generate such an image for each trajectory. Initially, we determine a minimum bounding rectangle (MBR) within a two-dimensional space, encapsulating all points of the whole trajectory dataset. Subsequently, the MBR is partitioned into equal-sized grids based on a predetermined hyperparameter width δ. Then for each trajectory T, its coordinates are mapped onto the grid, and each pixel within the grid cell is assigned a binary value, which is 1 if the trajectory point falls within the grid cell and 0 otherwise. Thus each raw trajectory is converted into a single-channel binary image BI_{2D}.

4.2 ConvTraj Network Structure

As shown in Figure 2, the ConvTraj consists of three submodules: 1D convolution, 2D convolution, and feature fusion. The 1D convolution extracts sequential features from the trajectory, while the 2D convolution captures its geo-distribution. The feature fusion module then combines these features for comprehensive analysis. Detailed descriptions of these submodules are provided below.

One-dimensional Convolution. As shown in Figure 2, 1D convolution is stacked by n residual blocks consisting of a 1D convolution layer, a non-linear ReLU layer, and a max-pooling layer. Each operation is performed on rows of Seq_{1D}. By default, the convolution kernel size is $2*3$, the number of channels is 32, the pooling stride is 2, and the number of stacking layers n is determined by the maximum length of the trajectory in the dataset. In the end, the features of all channels are flattened into a vector V_{1D}.

Two-dimensional Convolution. 2D convolution is also stacked by m residual blocks consisting of a 2D convolution layer, a non-linear ReLU layer, and an average-pooling layer. Each operation is performed on the single-channel binary image BI_{2D}. By default, the convolution kernel size is $3 * 3$, the number of channels is 4, the pooling stride is 2, and the number of stacking layers m is 4. In the end, the features of all channels are flattened into a vector V_{2D}.

Feature Fusion. After performing 1D and 2D convolution on the trajectory in parallel, we concatenate the resulting feature vectors

and pass them through an MLP. This submodule combines the sequence order features (V_{1D}) extracted by 1D convolution with the geo-distribution features (V_{2D}) extracted by 2D convolution, providing comprehensive information for similarity recognition. The final embedding V of the trajectory can be formalized as:

$$V = MLP([V_{1D}, V_{2D}]). \tag{1}$$

4.3 Training Pipeline

We employ the mainstream training pipeline as shown in Figure 3.

Loss Function. As shown in Figure 3, we use the combination of triplet loss [18, 28] L_T and MSE loss L_M as our loss function. i.e.:

$$Loss = L_T(T_a, T_p, T_n) + L_M(T_a, T_p, T_n), \tag{2}$$

where $L_T = max\{0, d(V_a, V_p) - d(V_a, V_n) - \eta\}$ and $L_M = |d(V_a, V_p) - f(T_a, T_p)| + |d(V_a, V_n) - f(T_a, T_n)|$. In which (T_a, T_p, T_n) is a triplet, and T_a is the anchor trajectory, T_p is the positive trajectory that has a smaller distance to T_a than the negative trajectory T_n. V_a, V_p and V_n are the high-dimensional vectors corresponding to T_a, T_p and T_n in the embedding space. $f(\cdot,\cdot)$ represents the true distance between trajectories, and $d(\cdot,\cdot)$ is the Euclidean distance [32, 33] between two vectors. Besides, η is the margin in the triplet loss whose value is $\eta = f(T_a, T_p) - f(T_a, T_n)$.

Triplet Selection Method. Many studies [11, 30, 32, 38] have proposed various strategies to select triplets for training, but these often bring additional training costs. In this paper, we use the simplest strategy to select triplets. We regard each trajectory in the training set as T_a in turn. For each T_a, we randomly select two trajectories from its top-k neighbors (k=200 by default) and use the trajectory closer to the T_a as T_p, and trajectory farther to T_a as T_n.

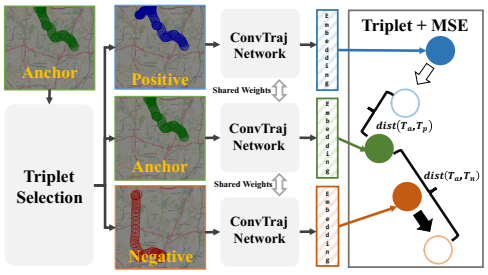


Figure 3: The training pipeline of ConvTraj.

5 THEORETICAL ANALYSIS

In this section, we will conduct some theoretical analysis to help justify why such a simple ConvTraj can work well. We take the DFD, which is widely used for trajectory similarity [26, 29, 36, 39], as an example for analysis. In summary, we found that: (1) After 1D max-pooling, the DFD value has almost no change. (2) For a randomly initialized kernel of 1D convolution, the DFD between two trajectories can still be maintained to a large extent. (3) Trajectories located in distant areas not only have a large DFD value but also have a large Euclidean distance through 2D convolution. Since 1D convolution essentially rotates and scales the trajectories and 2D convolution captures its geo-distribution, thus similar conclusions can be easily generalized to other measurements. Due to space limitations, we will only provide the proof of 1D max-pooling in the following, other proofs can be found here[2]. Basically, the analysis shows that 1D convolution and max-pooling can preserve the bounds for trajectory similarity learning, while 2D convolution can help capture the geo-distribution. This implies CNNs are a good choice in scenarios where trajectories need to be reduced in dimension or geo-distribution is required. This does not mean that RNNs or Transformers lack it, it is just difficult to analyze.

5.1 Discrete Frechet Distance

We first present the formal definition of Discrete Frechet Distance:

Definition 3 (Trajectory Coupling). A coupling L between two trajectories $T_1 = [p_1, p_2, ..., p_n]$ and $T_2 = [q_1, q_2, ..., q_m]$ is such a sequence of alignment:

$$L = (p_{a_1}, q_{b_1}), (p_{a_2}, q_{b_2}), ..., (p_{a_t}, q_{b_t}),$$

where $a_1 = 1, b_1 = 1, a_t = n, b_t = m$. For all $i = 1, ..., t$, we have $a_{i+1} = a_i$ or $a_{i+1} = a_i + 1$, and $b_{i+1} = b_i$ or $b_{i+1} = b_i + 1$.

Definition 4 (Discrete Frechet Distance). Given two trajectories $T_1 = [p_1, p_2, ..., p_n]$ and $T_2 = [q_1, q_2, ..., q_m]$, the Discrete Frechet Distance d_F between these two trajectories is:

$$d_F(T_1, T_2) = \min_{L}\{\max_{(p_i,q_j)\in L} d(p_i, q_j)\},$$

where L is an instance of coupling between T_1 and T_2, and $d(\cdot,\cdot)$ is Euclidean distance between two points.

5.2 One-dimensional Max-Pooling

Theorem 5.1 (One-dimensional Max-Pooling Bound). Given two sequences $X = [x_1, ..., x_M]$, $Y = [y_1, ..., y_N]$, and each $x_i \in X (y_i \in Y)$ is a l-dimensional vector, i.e., $x_i = [x_{i,1}, ..., x_{i,l}]^\mathsf{T}$. A one-dimensional max pooling operation $P(\cdot)$ on X, Y with size k and stride k, assuming that M and N are divisible by k. Then the following holds:

$$d_F(X, Y) - bound \le d_F(P(X), P(Y)) \le d_F(X, Y) + bound,$$

in which

$$bound = max\{d(X_i^{\downarrow}, X_i^{\uparrow})|1 \le i \le \frac{M}{k}\} + max\{d(Y_i^{\downarrow}, Y_i^{\uparrow})|1 \le i \le \frac{N}{k}\},$$

and $X_i^{\downarrow} = [x_{i,1}^{\downarrow}, ..., x_{i,l}^{\downarrow}]^\mathsf{T}$, $x_{i,j}^{\downarrow} = min\{x_{t,j}|t \in [(i-1)*k+1, i*k)\}$; $X_i^{\uparrow} = [x_{i,1}^{\uparrow}, ..., x_{i,l}^{\uparrow}]^\mathsf{T}$, $x_{i,j}^{\uparrow} = max\{x_{t,j}|t \in [(i-1)*k+1, i*k)\}$.(The same goes for $Y_i^{\downarrow}$ and $Y_i^{\uparrow}$)

Proof. Based on the triangle inequality of DFD, we can get:

$$\begin{aligned} d_F(X, Y) &\le d_F(X, P(X)) + d_F(P(X), Y) \\ &\le d_F(X, P(X)) + d_F(P(X), P(Y)) + d_F(P(Y), Y). \end{aligned}$$

Using this property again, we have:

$$d_F(P(X), P(Y)) \le d_F(X, Y) + (d_F(P(X), X) + d_F(Y, P(Y))).$$

Rearrange these two inequalities, we can get:

$$bound = d_F(X, P(X)) + d_F(Y, P(Y)).$$

Suppose $P(X) = [x_1^p, ..., x_{\frac{M}{k}}^p]$, and each x_i^p is a l-dimensional vector, i.e., $x_i^p = [x_{i,1}^p, ..., x_{i,l}^p]^\mathsf{T}$, where $x_{i,j}^p = max\{x_{t,j}|t \in [(i-1)*k+$

[2]https://arxiv.org/abs/2405.19761

$1, i * k)\}$. Then for $d_F(X, P(X))$, we can always construct such a coupling $L^* = \underbrace{(x_1, x_1^p), ..., (x_k, x_1^p)}_{\text{k}}, ..., \underbrace{(x_{M-k+1}, x_{\frac{M}{k}}^p), ..., (x_M, x_{\frac{M}{k}}^p)}_{\text{k}}$.

Thus $d_F(X, P(X)) \leq max_{(x_i, x_j^p) \in L^*} d(x_i, x_j^p)$. In this way, we divide the coupling L^* into $\frac{M}{k}$ groups. Without loss of generality, we take out the t-th group, that is $(x_{(t-1)*k+1}, x_t^p), ..., (x_{t*k}, x_t^p)$. Thus for $i \in [(t-1) * k + 1, t * k)$, we have:

$$\begin{aligned} max(d(x_i, x_t^p)) =& max(d([x_{i,1}, ..., x_{i,l}]^\top, [x_{t,1}^p, ..., x_{t,l}^p]^\top)) \\ =& max(d([x_{i,1}, ..., x_{i,l}]^\top, X_t^{\uparrow})) \\ \leq& d(X_t^{\downarrow}, X_t^{\uparrow}). \end{aligned}$$

Using this bound to $d_F(Y, P(Y))$ completes the proof. ∎

To verify the effectiveness of Theorem 5.1, we randomly selected 5000 pairs of trajectories from the Porto dataset for testing. The size and stride of max-pooling are set to 2, i.e., $k = 2$. As shown in Figure 4a, the DFD between the trajectories after max-pooling can accurately fall between the bounds predicted by Theorem 5.1. In addition, Figure 4b shows that the real DFD value has almost no change compared with the DFD after max-pooling, this implies that *max-pooling is a suitable technique that can reduce the dimensionality of trajectory sequences with almost no loss of effective features that are important for DFD-based similarity recognition.*

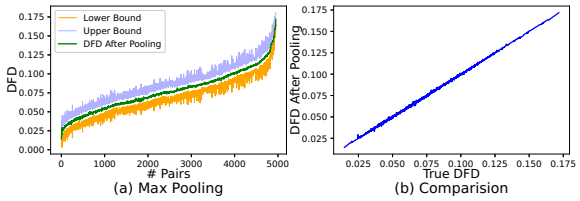


Figure 4: 1D max-pooling bound visualization on Porto.

6 EXPERIMENTS

6.1 Experimental Setting

Datasets. We evaluate ConvTraj on four widely used real-world datasets: **Geolife**[3], **Porto**[4], **Chengdu** and **TrajCL-Porto**[5]. For Geolife and Porto, we preprocess them using the method in [32], i.e. selecting trajectories in the central area of the city and removing items with less than 10 records. For TrajCL-Porto, it's an open-source dataset of TrajCL[7], we thus do not perform any processing. For Chengdu, we randomly selected 5000 trajectories from this dataset. The properties of these datasets are shown in Table 2.

Baselines. When we test on Geolife, Porto, and Chengdu, we follow existing works [10, 33] and compare ConvTraj with six methods, including **t2vec [21]** and **TrjSR [5]** based on self-supervised learning; **NeuTraj [32]**, **Traj2SimVec [38]**, **TrajGAT [33]**, and **TrajCL [7]** based on supervised learning. For the self-supervised method, since its goal is not to approximate the measurements, we thus perform the following steps to handle it. We first randomly select a part of the trajectory for pre-training (We select 10000 trajectories for Geolife and 200000 for Porto. In addition, we will also use these data to pre-train TrajCL). Then we add an MLP in the end and fine-tune it with the triplet selection method and loss function in subsection 4.3. For those methods which have open-source code [5, 7, 21, 32, 33], we directly use their implementation. For others [38], we implement it based on the settings of its paper. In addition, since many baselines have been evaluated on the TrajCL-Porto and the results have been reported in [7], we will directly compare our results with those of other baselines reported in [7].

Table 2: Trajectory Dataset Properties

Dataset	Geolife	Porto	Chengdu	TrajCL-Porto
# Total Items	13386	1601579	5000	9000
# Training Items	3000	3000	1000	7000
# Query Items	1000	500	1000	2000
# Candidate Items	9386	1598079	3000	2000
Avg-(# Points)	437.80	48.91	228.44	49.72
Min-(# Points)	11	11	29	20
Max-(# Points)	7579	3836	1575	200
Lat-Lon Area	(116.20, 116.50) (39.85, 40.07)	(-8.73, -8.50) (41.10, 41.24)	(104.04, 114.10) (30.65, 30.73)	(-8.70, -8.52) (41.10, 41.208)

Metrics. We follow existing works [32, 33, 38] and evaluate the effectiveness of these methods using the task of k nearest neighbor search. Specifically, we first use the top-k hitting rate (HR@k), which is the overlap percentage of detection top-k results with the ground truth. The second is the top-50 recall of the top-10 ground truth (R10@50), i.e. how many top 10 ground truths are recovered by the generated top 50 lists. These metrics can effectively evaluate whether the distance order in the embedding space is still preserved.

Implementation Details. We set the MLP output dimension in 1D preprocessing to 16. As the grid width δ decreases, ConvTraj will perform better, but the training cost will also increase [6], we thus set δ as 250 meters when generating images. For the Geolife, the number of residual blocks for 1D CNN is $n = 12(\lfloor \log_2 7579 \rfloor)$, Porto is $n = 11(\lfloor \log_2 3836 \rfloor)$, and TrajCl-Porto is $n = 7(\lfloor \log_2 200 \rfloor)$. We set the batch size to 128, the learning rate to 0.001, and the embedding size to 128. We evaluated Hausdorff, DFD, DTW, and EDR on Geolife and Porto, and evaluated Hausdorff, DFD, EDwP [24], and EDR on TrajCL-Porto. For each measurement on Geolife and Porto, we select three random seeds to repeat the experiment and report its average and variance. All experiments are conducted on a machine equipped with 36 CPU cores (Intel Core i9-10980XE CPU with 3.00GHz), 256 GB RAM, and a GeForce RTX 3090Ti GPU.

6.2 Effectiveness

Table 3, Table 4, and Table 5 present an overview of the performance exhibited by different methods concerning the top-k similarity search task on Geolife, Porto, and Chengdu, we can observe that: (1) On all datasets, ConvTraj significantly outperforms all methods on all metrics. Taking the Hausdorff distance on the Geolife as an example, compared with the state-of-the-art baseline NeuTraj,

[3]https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/
[4]https://www.kaggle.com/competitions/pkdd-15-predict-taxi-service-trajectory-i/data
[5]https://github.com/changyanchuan/TrajCL
[6]https://arxiv.org/abs/2405.19761

Table 3: Embedding Results On Geolife dataset (3 runs)

Geolife												
	Hausdorff			DFD			DTW			EDR		
Model	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50
t2vec	$22.82_{\pm 0.6}$	$24.48_{\pm 0.4}$	$44.60_{\pm 0.1}$	$26.36_{\pm 0.2}$	$28.33_{\pm 0.4}$	$53.45_{\pm 0.1}$	$26.70_{\pm 0.7}$	$28.91_{\pm 0.4}$	$55.40_{\pm 0.2}$	$18.34_{\pm 0.2}$	$20.95_{\pm 0.4}$	$46.22_{\pm 0.1}$
TrjSR	$34.86_{\pm 0.1}$	$37.26_{\pm 0.1}$	$66.56_{\pm 0.0}$	$29.92_{\pm 0.3}$	$33.29_{\pm 0.0}$	$61.62_{\pm 0.0}$	$32.64_{\pm 0.2}$	$36.52_{\pm 0.0}$	$67.91_{\pm 0.0}$	$18.26_{\pm 0.4}$	$19.83_{\pm 0.1}$	$45.52_{\pm 0.0}$
TrajCL	$31.08_{\pm 0.0}$	$37.21_{\pm 0.0}$	$72.55_{\pm 0.0}$	$33.51_{\pm 0.1}$	$38.98_{\pm 0.3}$	$75.82_{\pm 0.4}$	$12.48_{\pm 0.1}$	$14.63_{\pm 0.0}$	$32.05_{\pm 0.0}$	$20.33_{\pm 0.1}$	$22.74_{\pm 0.0}$	$44.90_{\pm 0.3}$
NeuTraj	$42.31_{\pm 0.1}$	$48.40_{\pm 0.1}$	$80.38_{\pm 0.0}$	$58.50_{\pm 0.1}$	$64.47_{\pm 0.4}$	$94.44_{\pm 0.4}$	$30.57_{\pm 0.0}$	$33.68_{\pm 0.2}$	$62.87_{\pm 0.5}$	$11.98_{\pm 16.4}$	$14.28_{\pm 19.8}$	$21.63_{\pm 10.9}$
Traj2SimVec	$33.49_{\pm 0.9}$	$42.39_{\pm 0.3}$	$65.12_{\pm 0.2}$	$40.39_{\pm 0.9}$	$42.75_{\pm 0.3}$	$70.20_{\pm 0.0}$	$13.24_{\pm 0.4}$	$15.39_{\pm 0.7}$	$35.09_{\pm 0.0}$	$10.44_{\pm 0.6}$	$12.39_{\pm 0.7}$	$18.45_{\pm 0.0}$
TrajGAT	$25.80_{\pm 1.2}$	$30.57_{\pm 0.7}$	$66.69_{\pm 0.2}$	$21.20_{\pm 0.9}$	$25.87_{\pm 1.2}$	$61.91_{\pm 0.9}$	$19.98_{\pm 0.2}$	$24.77_{\pm 0.4}$	$59.36_{\pm 0.2}$	$15.58_{\pm 0.7}$	$17.84_{\pm 1.3}$	$36.78_{\pm 0.2}$
ConvTraj	$\mathbf{57.73}_{\pm 0.2}$	$\mathbf{63.69}_{\pm 0.3}$	$\mathbf{95.20}_{\pm 0.0}$	$\mathbf{62.73}_{\pm 0.5}$	$\mathbf{68.86}_{\pm 0.3}$	$\mathbf{97.34}_{\pm 0.0}$	$\mathbf{41.56}_{\pm 0.3}$	$\mathbf{46.46}_{\pm 0.6}$	$\mathbf{83.70}_{\pm 0.0}$	$\mathbf{26.95}_{\pm 2.5}$	$\mathbf{28.64}_{\pm 2.9}$	$\mathbf{54.93}_{\pm 4.9}$
Gap with SOTA	+15.42	+15.29	+14.82	+4.23	+4.39	+2.90	+8.92	+9.94	+15.79	+6.62	+5.90	+8.71

Table 4: Embedding Results On Porto dataset (3 runs)

Porto												
	Hausdorff			DFD			DTW			EDR		
Model	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50
t2vec	$5.88_{\pm 0.8}$	$7.28_{\pm 0.8}$	$17.08_{\pm 0.0}$	$6.28_{\pm 0.9}$	$7.66_{\pm 0.8}$	$17.84_{\pm 0.0}$	$8.16_{\pm 0.7}$	$9.60_{\pm 0.6}$	$20.92_{\pm 0.1}$	$4.24_{\pm 0.9}$	$4.76_{\pm 0.6}$	$12.30_{\pm 0.2}$
TrjSR	$13.26_{\pm 0.7}$	$14.79_{\pm 0.2}$	$33.46_{\pm 0.0}$	$10.36_{\pm 0.3}$	$14.26_{\pm 0.6}$	$37.48_{\pm 0.1}$	$15.44_{\pm 0.3}$	$18.29_{\pm 0.4}$	$38.23_{\pm 0.1}$	$6.13_{\pm 0.1}$	$8.42_{\pm 0.1}$	$23.20_{\pm 0.5}$
TrajCL	$12.55_{\pm 0.1}$	$15.37_{\pm 0.1}$	$35.37_{\pm 0.0}$	$14.08_{\pm 0.1}$	$18.01_{\pm 0.0}$	$42.97_{\pm 0.1}$	$1.51_{\pm 0.0}$	$2.33_{\pm 0.0}$	$6.96_{\pm 0.9}$	$7.99_{\pm 0.0}$	$10.47_{\pm 0.1}$	$25.53_{\pm 0.5}$
NeuTraj	$19.32_{\pm 0.0}$	$24.07_{\pm 0.0}$	$51.43_{\pm 0.0}$	$27.71_{\pm 0.1}$	$33.64_{\pm 0.0}$	$66.58_{\pm 0.1}$	$13.48_{\pm 0.0}$	$16.33_{\pm 0.0}$	$34.81_{\pm 0.1}$	$2.52_{\pm 0.1}$	$3.79_{\pm 0.1}$	$10.83_{\pm 0.1}$
Traj2SimVec	$14.33_{\pm 0.9}$	$16.32_{\pm 0.2}$	$37.45_{\pm 0.0}$	$16.37_{\pm 0.3}$	$20.03_{\pm 0.0}$	$44.11_{\pm 0.2}$	$10.44_{\pm 0.3}$	$13.22_{\pm 0.3}$	$18.30_{\pm 0.0}$	$1.59_{\pm 0.3}$	$1.85_{\pm 0.1}$	$7.33_{\pm 0.2}$
TrajGAT	$16.48_{\pm 0.8}$	$18.29_{\pm 0.6}$	$43.58_{\pm 0.3}$	$13.49_{\pm 0.8}$	$20.38_{\pm 0.4}$	$39.78_{\pm 0.2}$	$11.02_{\pm 0.2}$	$12.58_{\pm 0.0}$	$20.79_{\pm 0.0}$	$4.78_{\pm 0.6}$	$6.23_{\pm 0.0}$	$12.34_{\pm 0.0}$
ConvTraj	$\mathbf{27.68}_{\pm 0.0}$	$\mathbf{33.27}_{\pm 0.1}$	$\mathbf{67.20}_{\pm 0.0}$	$\mathbf{34.97}_{\pm 0.2}$	$\mathbf{40.59}_{\pm 0.0}$	$\mathbf{77.33}_{\pm 0.5}$	$\mathbf{19.77}_{\pm 0.1}$	$\mathbf{24.46}_{\pm 0.4}$	$\mathbf{52.83}_{\pm 0.0}$	$\mathbf{13.84}_{\pm 0.5}$	$\mathbf{15.61}_{\pm 0.3}$	$\mathbf{37.00}_{\pm 2.8}$
Gap with SOTA	+8.36	+9.20	+15.77	+7.26	+6.95	+10.75	+4.33	+6.17	+14.60	+5.85	+5.14	+11.47

Table 5: Embedding Results On Chengdu dataset

Chengdu												
	Hausdorff			DFD			DTW			EDR		
Model	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50	HR@5	HR@10	R10@50
t2vec	16.16	16.52	30.69	21.34	22.34	44.06	21.00	22.68	46.03	14.18	16.37	37.41
TrjSR	10.44	12.09	25.44	11.08	12.77	25.96	16.42	18.32	36.63	18.80	19.78	40.02
TrajCL	22.14	27.04	61.93	19.00	21.42	51.79	23.48	27.20	59.89	17.40	20.53	48.58
NeuTraj	21.82	23.27	39.09	32.28	34.70	49.51	28.40	29.33	46.90	10.44	11.54	24.65
Traj2SimVec	18.34	20.17	43.67	25.16	28.33	42.78	18.66	19.37	40.69	6.93	8.31	19.18
TrajGAT	19.98	25.26	57.57	16.24	19.11	49.18	23.96	28.32	65.41	17.94	20.55	48.40
ConvTraj	**36.26**	**42.78**	**76.67**	**53.34**	**58.18**	**93.14**	**34.90**	**40.40**	**76.23**	**21.50**	**25.34**	**55.21**
Gap with SOTA	+14.12	+15.74	+19.10	+21.06	+23.48	+41.45	+6.50	+11.07	+10.82	+2.70	+4.81	+6.63

ConvTraj exceeds by more than 11% in all metrics, with the largest improvement of 15.42% for HR@5 and the smallest improvement of 11.64% for HR@1. In addition, even for the Porto which contains 1.6 million trajectories, R10@50 has at least a 10.75% improvement on four measurements. This non-negligible improvement in performance is impressive given the fact that the sequence order features extracted by 1D convolution and the geographical distribution of the trajectory extracted by 2D convolution are both very beneficial to generating high-quality trajectory embedding representations. (2) The advantage of ConvTraj is evident in all measurements, which shows that ConvTraj is a general framework for different measurements. We can observe that no method can handle all measurements well. For example, NeuTraj performs best on the Hausdorff and DFD, while TrjSR and TrajCL have advantages on DTW and EDR respectively, which is also mutually verified with the results in [10]. However, ConvTraj achieves state-of-the-art accuracy in all measurements. Compared to the state-of-the-art, ConvTraj achieves an average improvement of 10.22%, 8.02%, 7.59%, and 7.03% on all metrics of the Hausdorff, DFD, DTW, and EDR in Porto respectively. (3) We also noticed that compared with the results on the Geolife and Porto datasets, the TrajGAT method performed better on the Chengdu dataset. This may be because the longitude and latitude of the Chengdu dataset cover a larger area, so the quadtree-based modeling method of the TrajGAT is more effective.

Table 6 presents the experimental results on TrajCL-Porto, we can observe that: (1) Similar to its performance on the Geolife and Porto, the ConvTraj method surpasses state-of-the-art in almost all metrics for four measurements. Compared with the state-of-the-art, ConvTraj achieves improvements of 12%, 23%, 6.8%, and 14.2% on the HR@5 metrics of EDR, EDwP, Hausdorff, and DFD. (2) Even though both were tested on the Porto dataset, the performance gap between Table 4 and Table 6 is very large. For example, the HR@5 of the TrajCL and ConvTraj in Table 6 on the DFD are 0.618 and 0.749 respectively, but in Table 4 they are 0.141 and 0.349 respectively. The

Table 6: Embedding Results On TrajCL-Porto dataset

TrajCL-Porto												
	EDR			EDwP			Hausdorff			DFD		
Model	HR@5	HR@20	R5@20	HR@5	HR@20	R5@20	HR@5	HR@20	R5@20	HR@5	HR@20	R5@20
t2vec	0.125	0.164	0.286	0.399	0.518	0.751	0.405	0.549	0.770	0.504	0.651	0.883
TrjSR	0.137	0.147	0.273	0.271	0.346	0.535	0.541	0.638	0.880	0.271	0.356	0.523
E2DTC [15]	0.122	0.157	0.272	0.390	0.514	0.742	0.391	0.537	0.753	0.498	0.648	0.879
CSTRM [23]	0.138	0.191	0.321	0.415	0.536	0.753	0.459	0.584	0.813	0.421	0.557	0.768
TrajCL	0.172	**0.222**	0.376	0.546	0.646	0.881	0.643	0.721	0.954	0.618	0.740	0.955
T3S [31]	0.140	0.192	0.325	0.377	0.498	0.702	0.329	0.482	0.668	0.595	0.728	0.946
Traj2SimVec	0.119	0.163	0.285	0.172	0.253	0.390	0.339	0.429	0.543	0.529	0.664	0.894
TrajGAT	0.090	0.102	0.184	0.201	0.274	0.469	0.686	0.740	0.969	0.362	0.403	0.704
ConvTraj	**0.292**	0.181	**0.414**	**0.776**	**0.826**	**0.987**	**0.754**	**0.770**	**0.983**	**0.760**	**0.786**	**0.984**
Gap with SOTA	+0.12	−0.041	+0.038	+0.23	+0.18	+0.106	+0.068	+0.03	+0.014	+0.142	+0.046	+0.029
ConvTraj-1D CNN	0.230	0.097	0.279	0.648	0.685	0.937	0.732	0.757	0.983	0.736	0.769	0.978
ConvTraj-2D CNN	0.285	0.174	0.387	0.611	0.586	0.949	0.746	0.769	0.983	0.565	0.528	0.908

Table 7: Efficiency Comparison

Method	# Paras	Geolife				Porto			
		Pre-trained time t_{epoch} * (# epoch)	Train time t_{epoch} * (# epoch)	Train time Per Epoch	Inference time	Pre-trained time t_{epoch} * (# epoch)	Train time t_{epoch} * (# epoch)	Train time Per Epoch	Inference time
t2vec	2.86M	17.97s * 10	0.27s * 200	18.24s	0.89s	328.12s * 10	0.27s * 200	328.39s	61.64s
TrjSR	≈ 40000	273.05s * 3	0.27s * 200	273.33s	**0.09**s	11800s * 3	0.27s * 200	11800.27s	**11.69**s
TrajCL	5.49M	14.03s * 54	145.73s * 30	159.76s	11.42s	208.73s * 75	52.14s * 30	260.87s	367.12s
NeuTraj	0.10M	-	149.13s * 100	149.13s	41.48s	-	230.29s * 100	230.29s	832.58s
TrajGAT	11.45M	-	2613s * 50	2613s	257.49s	-	1843s * 50	1843s	4946.38s
ConvTraj	0.13M	-	1.57s * 200	**1.57**s	0.41s	-	1.07s * 200	**1.07**s	28.53s

reason is that the TrajCL-Porto dataset contains fewer trajectories. When performing the top-k similarity search task, the TrajCL-Porto dataset only has 2000 candidate trajectories. However, the Porto used in Table 4 contains 1598079 candidate trajectories, which results in a more comprehensive result. (3) We also evaluate the performance of ConvTraj using only 1D convolution (ConvTraj-1D CNN) or 2D convolution (ConvTraj-2D CNN) on TrajCL-Porto, and we can observe that ConvTraj's performance degrades after missing some features, but still has excellent performance.

6.3 Efficiency

We evaluate the efficiency of all baselines with open-source code on Geolife and Porto and report the results in Table 7. We also report the pre-training time of methods that require pre-training.

As shown, compared to existing RNN-based and Transformer-based methods, ConvTraj not only has fewer parameters (only 0.03M more than NeuTraj) but also has great advantages in training and inference speed. Taking the Porto with 1.6 million items as an example, compared with the most efficient Transformer-based model TrajCL, the training speed per epoch and the inference speed of ConvTraj are at least 243.80x and 12.87x faster respectively. Compared with the most efficient RNN-based model t2vec, the training speed per epoch and the inference speed of ConvTraj are at least 306.91x and 2.16x faster respectively. The reason for such a huge improvement is that compared to Transformer-based methods, ConvTraj has fewer parameters. Meanwhile, compared with RNN-based methods, although the parameters of ConvTraj are relatively large, the training and inference of ConvTraj are more efficient due to the inherent low parallelism of RNN. We also note that: (1) Compared with the CNN-based TrjSR, ConvTraj has no advantage in inference, but the training is faster because TrjSR requires pre-training on a large number of trajectories, and only uses fewer layers during inference, which also shows the superiority of CNN in terms of efficiency. (2) Although t2vec and NeuTraj are based on RNN, and NeuTraj has fewer parameters, t2vec is more efficient. The reason is that NeuTraj needs to select more triplets during training and compute spatial attention based on adjacent grids at each time step.

6.4 Ablation Studies

6.4.1 The Role of 1D and 2D Convolution. Our ConvTraj combines 1D and 2D convolutions, we thus conducted the following experiments to evaluate the contributions of each module: (1) 1D CNN. Only using 1D convolution. (2) 2D CNN. Only using 2D convolution. (3) 1D+2D. Using 1D and 2D convolution together. The results in Table 8 show that for all measurements, neglecting any of these modules leads to a reduction in performance. In addition, we observe that 2D CNN outperforms most baselines, including TrjSR, which also uses 2D convolution. A similar conclusion can also be derived from Table 6. We explain that the goal of TrjSR is to reconstruct a high-resolution image from a low-resolution so that it can be as close as possible to the original image, thus the backbone and loss used are quite different from our 2D CNN. Furthermore, although we fine-tuned TrjSR, our 2D CNN is trained end-to-end and thus has more advantages.

Table 8: Ablation Studies Results: The Role of 1D and 2D Convolution

	Method	Hausdorff			DFD			DTW			EDR		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Geolife	1D CNN	38.89	56.90	79.01	62.51	76.44	95.50	32.02	45.82	68.34	11.32	14.60	16.09
	2D CNN	57.97	72.11	92.49	44.28	54.32	85.37	35.46	45.19	75.22	22.41	26.59	50.00
	1D+2D	**63.69**	**76.12**	**95.20**	**68.86**	**79.52**	**97.34**	**46.46**	**59.26**	**83.70**	**28.64**	**30.75**	**54.93**
Porto	1D CNN	13.80	25.53	39.68	10.30	17.86	32.02	3.16	8.24	13.06	12.58	14.45	24.86
	2D CNN	28.46	40.72	60.04	24.52	35.35	54.84	19.88	30.39	34.91	9.90	17.26	26.96
	1D+2D	**33.27**	**45.98**	**67.20**	**40.59**	**53.35**	**77.33**	**24.46**	**35.36**	**52.83**	**15.61**	**21.45**	**37.00**

6.4.2 Use LSTM to Replace 1D Convolution. The role of 1D convolution is to capture the sequential features. Although RNNs are commonly used for this purpose [32], we aim to show the important role of 1D convolution in ConvTraj by replacing it with LSTM. We will compare three methods to show its effectiveness in capturing sequential features: (1) 2D CNN. Only using 2D convolution. (2) LSTM+2D. Using LSTM and 2D convolution together. (3) 1D+2D. Using 1D and 2D convolution together. The dataset is the same as Table 1, called **Porto-S**, and the number of GPS points contained in each trajectory in Porto-S ranges from 104 to 888. In addition, we generated two more datasets, **Porto-S-10** and **Porto-S-70**, which contain the first 10 and 70 points of each trajectory in Porto-S respectively. We can observe that the performance of LSTM+2D and 1D+2D is similar in the Porto-S-10, and LSTM+2D even performs slightly better than 1D+2D at some measurements (e.g., DTW and EDR), and both methods are significantly outperform than 2D CNN. These results show that LSTM performs well in capturing sequential features when the trajectory contains fewer points. However, as the number of points in a trajectory increases, the ability of LSTM to capture sequential features gradually decreases. For example, the HR@1 of 1D+2D and LSTM+2D on Porto-S-10 are 62.40% and 63.20% respectively for DTW. However, the HR@1 on Porto-S-70 are 52.40% and 46.40% respectively, and on Porto-S they are 38.60% and 22.20%. The gaps between them are -0.80%, +6.0%, +16.4%, increasing progressively. Even LSTM+2D performs nearly as well as 2D CNN alone on Porto-S. These results show that RNNs struggle to capture the sequential features of trajectories with a large number of GPS points, whereas 1D CNNs do not exhibit this limitation.

6.4.3 2D Image Construction. We checked the image generation strategy used in TrjSR. We can find that when this strategy is directly applied to ConvTraj, the performance of the model degrades. Replacing the average pooling in the 2D convolution with max pooling brings performance close to ConvTraj. The reason is that for grayscale images, more points indicate a longer duration in a grid, allowing different pixel values to capture the temporal properties. Average pooling can introduce noise that hinders the model's ability to capture geo-distribution. In contrast, max pooling effectively extracts the strongest features from the grayscale image.

6.4.4 Loss Function. We conducted an ablation study on two loss functions on the Geolife and found that after removing the triplet loss, all metrics declined. However, after removing the MSE loss, the metrics of Hausdorff and DFD declined, while the metrics of DTW and EDR increased. We guess the reason is that the range of distance values of DTW and EDR is relatively large compared to Hausdorff and DFD, thus causing the MSE loss to encounter problems during scaling. A more detailed discussion may be studied in the future.

6.5 Training and Convergence Discussion

6.5.1 Motivation Experiment. Since the Transformer-based model has more parameters, we thus tested 200, 1000, and 2000 training epochs for the experiments in the introduction. We can find that even if the vanilla Transformer-based model reached convergence, it did not show an advantage over the 1D CNN that was only trained for 200 epochs. However, the training cost of the Transformer-based model was very high due to its larger number of parameters. We also evaluated the performance of each model after increasing the training trajectories from 3000 to 6000, and 10000. The results show that the vanilla Transformer-based model does not significantly outperform the 1D CNN even with increased training data.

6.5.2 The Training Details of TrajCL. Since the baseline of TrajCL performs best among all current Transformer-based baselines, we would like to add more details about the training and convergence of TrajCL. In our experiments, TrajCL's results are based on its default open-source settings for a fair comparison. For Geolife, we pre-train TrajCL with 10000 trajectories using the default training epoch of 100 in its open-source code, then fine-tune TrajCL with the ground truth, with the default training epoch of 30 in its open-source code. We can see that for Hausdorff and DFD, the model does not seem to have converged at this time, but for DTW and EDR, the model has overfitted. We thus increased training epochs from 30 to 100 and we can observe that the model has converged after 100 epochs. In addition, the performance of Hausdorff and DFD increases after 100 epochs of training, but the performance of DTW and EDR decreases after 100 epochs of training.

More detailed results and explanations can be found here[7].

7 CONCLUSION

This paper argues that trajectory similarity learning should pay more attention to local similarity and proposes a CNN-based framework ConvTraj. Some analysis is conducted to help justify its effectiveness and extensive experiments show its superiority.

ACKNOWLEDGMENTS

This work was sponsored by the Fundamental Research Funds for the Central Universities (226-2024-00145, 226-2024-00216), NSFC Grant No. 62402420, and Zhejiang University Education Foundation Qizhen Scholar Foundation.

[7] https://arxiv.org/abs/2405.19761

REFERENCES

[1] Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. 2018. Subtrajectory Clustering: Models and Algorithms. In *PODS*. ACM, 75–87.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. 1993. Efficient Similarity Search In Sequence Databases. In *FODO (Lecture Notes in Computer Science)*, Vol. 730. Springer, 69–84.

[3] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.* 5 (1995), 75–91.

[4] Stefan Atev, Grant Miller, and Nikolaos P. Papanikolopoulos. 2010. Clustering of Vehicle Trajectories. *IEEE Trans. Intell. Transp. Syst.* 11, 3 (2010), 647–657.

[5] Hanlin Cao, Haina Tang, Yulei Wu, Fei Wang, and Yongjun Xu. 2021. On Accurate Computation of Trajectory Similarity via Single Image Super-Resolution. In *IJCNN*. IEEE, 1–9.

[6] T.-H. Hubert Chan, Arnaud Guerquin, and Mauro Sozio. 2018. Fully Dynamic k-Center Clustering. In *WWW*. ACM, 579–587.

[7] Yanchuan Chang, Jianzhong Qi, Yuxuan Liang, and Egemen Tanin. 2023. Contrastive Trajectory Similarity Learning with Dual-Feature Attention. In *ICDE*. IEEE, 2933–2945.

[8] Yanchuan Chang, Jianzhong Qi, Egemen Tanin, Xingjun Ma, and Hanan Samet. 2021. Sub-trajectory Similarity Join with Obfuscation. In *SSDBM*. ACM, 181–192.

[9] Yanchuan Chang, Egemen Tanin, Xin Cao, and Jianzhong Qi. 2023. Spatial Structure-Aware Road Network Embedding via Graph Contrastive Learning. In *EDBT*. OpenProceedings.org, 144–156.

[10] Yanchuan Chang, Egemen Tanin, Gao Cong, Christian S. Jensen, and Jianzhong Qi. 2023. Trajectory Similarity Measurement: An Efficiency Perspective. *CoRR* abs/2311.00960 (2023).

[11] Zhihao Chang, Linzhu Yu, Yanchao Xu, and Wentao Hu. 2024. Neural Embeddings for kNN Search in Biological Sequence. In *AAAI*. AAAI Press, 38–45.

[12] Lei Chen and Raymond T. Ng. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*. Morgan Kaufmann, 792–803.

[13] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. ACM, 491–502.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*. ACL, 1724–1734.

[15] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. 2021. E^2DTC: An End to End Deep Trajectory Clustering Framework via Self-Training. In *ICDE*. IEEE, 696–707.

[16] Ziquan Fang, Yuntao Du, Xinjun Zhu, Danlei Hu, Lu Chen, Yunjun Gao, and Christian S. Jensen. 2022. Spatio-Temporal Trajectory Similarity Learning in Road Networks. In *KDD*. ACM, 347–356.

[17] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A Graph-based Approach for Trajectory Similarity Computation in Spatial Networks. In *KDD*. ACM, 556–564.

[18] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737* (2017).

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.

[20] Rikard Laxhammar and Göran Falkman. 2014. Online Learning and Sequential Anomaly Detection in Trajectories. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 6 (2014), 1158–1173.

[21] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. IEEE Computer Society, 617–628.

[22] Bin Lin and Jianwen Su. 2008. One Way Distance: For Shape Based Similarity Search of Moving Object Trajectories. *GeoInformatica* 12, 2 (2008), 117–142.

[23] Xiang Liu, Xiaoying Tan, Yuchun Guo, Yishuai Chen, and Zhe Zhang. 2022. CSTRM: Contrastive Self-Supervised Trajectory Representation Model for trajectory similarity computation. *Comput. Commun.* 185 (2022), 159–167.

[24] Sayan Ranu, Deepak P, Aditya D. Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*, Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman (Eds.). IEEE Computer Society, 999–1010.

[25] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *SIGMOD*. ACM, 725–740.

[26] Bo Tang, Man Lung Yiu, Kyriakos Mouratidis, and Kai Wang. 2017. Efficient Motif Discovery in Spatial Trajectories Using Discrete Fréchet Distance. In *EDBT*. OpenProceedings.org, 378–389.

[27] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Discovering Similar Multidimensional Trajectories. In *ICDE*, Rakesh Agrawal and Klaus R. Dittrich (Eds.). IEEE Computer Society, 673–684.

[28] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research* 10, 2 (2009).

[29] Dong Xie, Feifei Li, and Jeff M. Phillips. 2017. Distributed Trajectory Similarity Search. *Proc. VLDB Endow.* 10, 11 (2017), 1478–1489.

[30] Peilun Yang, Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Wenjie Zhang. 2022. TMN: Trajectory Matching Networks for Predicting Similarity. In *ICDE*. IEEE, 1700–1713.

[31] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *ICDE*. IEEE, 2183–2188.

[32] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *ICDE*. IEEE, 1358–1369.

[33] Di Yao, Haonan Hu, Lun Du, Gao Cong, Shi Han, and Jingping Bi. 2022. TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation. In *SIGKDD*. ACM, 2275–2285.

[34] Di Yao, Chao Zhang, Zhihua Zhu, Qin Hu, Zheng Wang, Jian-Hui Huang, and Jingping Bi. 2018. Learning deep representation for trajectory clustering. *Expert Syst. J. Knowl. Eng.* 35, 2 (2018).

[35] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*. IEEE Computer Society, 201–208.

[36] Dongxiang Zhang, Zhihao Chang, Sai Wu, Ye Yuan, Kian-Lee Tan, and Gang Chen. 2022. Continuous Trajectory Similarity Search for Online Outlier Detection. *IEEE Trans. Knowl. Data Eng.* 34, 10 (2022), 4690–4704.

[37] Dongxiang Zhang, Zhihao Chang, Dingyu Yang, Dongsheng Li, Kian-Lee Tan, Ke Chen, and Gang Chen. 2023. SQUID: subtrajectory query in trillion-scale GPS database. *VLDB J.* 32, 4 (2023), 887–904.

[38] Hanyuan Zhang, Xinyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*. ijcai.org, 3209–3215.

[39] Jiahao Zhang, Bo Tang, and Man Lung Yiu. 2019. Fast Trajectory Range Query with Discrete Frechet Distance. In *EDBT*. OpenProceedings.org, 634–637.

[40] Silin Zhou, Peng Han, Di Yao, Lisi Chen, and Xiangliang Zhang. 2023. Spatial-temporal fusion graph framework for trajectory similarity computation. *WWW* 26, 4 (2023), 1501–1523.

[41] Silin Zhou, Jing Li, Hao Wang, Shuo Shang, and Peng Han. 2023. GRLSTM: Trajectory Similarity Computation with Graph-Based Residual LSTM. In *AAAI*. AAAI Press, 4972–4980.