On Efficient Computation of Trajectory Similarity via Single Image Super-Resolution

Hanlin Cao[1], Haina Tang[1*], Yulei Wu[2], Fei Wang,[3] Yongjun Xu[3]
[1]*Department of Artificial Intelligence*, *University of Chinese Academy of Science*, *Beijing, China*
[2]*Department of Computer Science*, *University of Exeter*, *Exeter, United Kingdom*
[3]*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing, China*
caohanlin18@mails.ucas.ac.cn, hntang@ucas.ac.cn, Y.L.Wu@exeter.ac.uk, {wangfei,xyj}@ict.ac.cn

***Abstract*—Measuring the similarity between trajectories is important to many location-aware applications. However, the trajectory data collected in the real world suffer from the low-quality problem caused by non-uniform sampling rates and noises, which significantly affects the accuracy of similarity measurement. Traditional pairwise point-matching methods are susceptible to non-uniform sampling rates inherently, since they assume the consistent sampling rate. Although the recurrent neural network (RNN) based methods have addressed this problem by complementing trajectory data, they have the drawback of predicting positions conditioned on the historical data generated by the model itself, which could lead to accumulated bias during the inference stage. In this paper, we propose a novel generative model to address the important issue of low-quality trajectory data based on single image super-resolution. The trajectory similarity is thus computed using trajectory images instead of the trajectory sequential data. By utilizing the image to represent trajectory, we effectively overcome the difficulty in complementing trajectory data encountered by traditional methods. Extensive experimental results using real-world datasets demonstrate that our method outperforms existing methods in terms of both accuracy and efficiency.**

I. Introduction

The advent of the ubiquitous mobile and IoT devices has produced torrents of trajectory data, and these data in turn has backed up a wide range of location-aware applications, such as recommendation, ridesourcing, navigation, to name a few. Trajectory data mining has received considerable attentions, which aims to extract trajectory patterns and understand user behaviors from the raw trajectory data, so that novel and personalized services can be provided. Among all the trajectory data mining techniques, computing the similarity between trajectories has always been recognized as a fundamental research, which supports many trajectory based applications, such as moving together discovery, trajectory-user linking, trajectory clustering and trajectory anomaly detection.

Computing trajectory similarity is challenging in practice due to the longstanding issues of the low-quality trajectory data in reality. The raw trajectory data usually consist of ordered sequences of coordinate-timestamp tuple, which is collected by the sensors deployed on the moving objects that can periodically transmit the location of the object over time. The problem of low-quality trajectory data mainly manifests in two aspects:

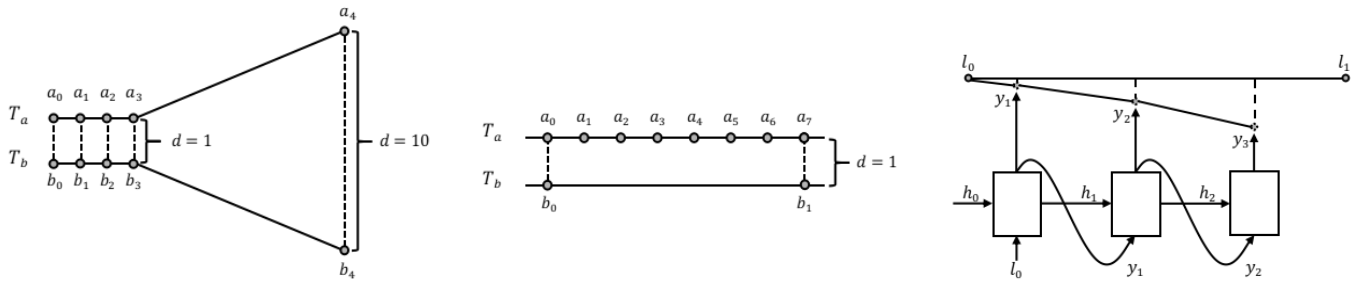
1. **Non-uniform sampling rates.** The sampling rates often vary across different trajectories due to the variable device settings and communication conditions. Even for the same trajectory, the sampling rates may vary in different segments. This could bring about the mixed dense and sparse segments in trajectory data. Besides, the trajectories collected from the social networks like geo-tagged tweets and geo-tagged photo albums are inherently non-uniform [1].
2. **Noise.** The raw trajectory data always come with noises in reality, which indicates that the collected data have a certain degree of deviation from the real trajectory. Such noises could result from GPS satellite geometry, signal blockage, atmospheric conditions, and receiver design features/quality [2].

The conventional methods of measuring trajectory similarity, including Dynamic Time Warping (DTW) [3], Longest Common Subsequences (LCSS) [4] and Edit Distance on Real sequence (EDR) [5], are mainly based on pairwise point-matching, namely, the similarity is measured by some kind of aggregation of the distances between trajectory sampling points. Such kind of methods may encounter performance degradation when being used to process noisy trajectory data. That is because the originally similar trajectories will become less similar depending on the degree of noises. Besides, pairwise point-matching based methods have difficulties in dealing with the problem of non-uniform trajectory sampling rates.

Let us take EDR as an example to illustrate the problem of non-uniform sampling rates. EDR will assign a distance of 0 to points pair within the distance threshold ϵ, otherwise assigns an edit distance of 1. Given $\epsilon = 1$, in Fig. 1a, EDR will only yield a distance of 1 to (a_4, b_4) and the remaining points pairs will be matched with the distance of 0. However, T_a and T_b are unlike in most part. Therefore, the distance of 1 does not reflect the actual dissimilarity between the two trajectories. Similarly, as illustrated in Fig. 1b, only (a_0, b_0) and (a_7, b_1) can be matched, while the remaining unmatched points will yield a distance of 6 by EDR. Although such two trajectories are similar, they will be assumed dissimilar using EDR due to the different sampling rates.

Recently, the recurrent neural network (RNN) based methods were proposed to cope with the above problems by managing to predict the missing and accurate positions through maximizing

* Corresponding author.



(a) (b) (c)

Fig. 1: Illustration of the challenges in computing trajectory similarity. (a) Originally dissimilar trajectories are assumed similar by EDR; (b) Originally similar trajectories are assumed dissimilar by EDR; (c) The prediction process of RNN cannot take advantage of the information from the near future.

the likelihood of each target position. However, despite their opportunity, the RNN based methods still suffer from the so-called *exposure bias* in the inference stage [6]: the model predicts the next trajectory point conditioned on the previously predicted ones that may be never seen in the training data. As a result, the prediction error would be accumulated along with the generated trajectory sequence. Besides, the data point in a trajectory depends not only on the preceding (historical) points but also the succeeding (future) points within a segment of the trajectory, while the RNN based methods only give inferences based on historical data points. Fig. 1c illustrates an example, where the RNN predicts y_3 based on the information contained in y_2 and h_2, while it cannot make use of the information from the near future, like l_1.

Moreover, both pairwise point-matching based and RNN based methods heavily rely on the sequential order, thus they all fall short in directly distinguishing similar trajectories with inverse orders. For example, a person's daily commuting trajectories, including going to work from home and going back home after work, should be regarded similar because they match the same route. With the aforementioned existing methods, we need to double the computing time if we want to check whether they belong to inversely-ordered similar trajectories.

The single image super-resolution (SISR) refers to the task of estimating a high-resolution (HR) image from a single low-resolution (LR) counterpart. By super-resolving, the image details can be reconstructed and thus the quality of the image is enhanced. In practice, the LR images are usually downsampled from the HR images with some blurring by adding noises, which is very similar to the situation that the low-quality trajectories result from the non-uniform sampling rates (downsample) and noises (blurring). This observation enlightens us to introduce SISR to improve the quality of trajectory data. Although powerful deep learning models have been applied to SISR and have achieved state-of-the-art performance, there are still challenges when applying them to real-world scenarios. First, current deep learning models often have massive parameters to recover photo-realistic images, and they need large-scale data to fit the models. It is therefore computational expensive to adopt them to simple scenarios like trajectory, license plate and text. A lighter deep learning model with fewer parameters and little performance degradation is expected to be developed for our research. Second, the commonly used optimization target of super-resolution (SR) algorithms is the minimization of the mean squared error (MSE) between the recovered HR image and the ground truth. However, the optimization objectives should be specific for different applications. For example, in our research, we aim to reconstruct the images for trajectory similarity computation, where simply minimizing MSE does not necessarily benefit similarity measurement and we demonstrate it in Section V-D.

In this paper, we propose a novel generative model for the computation of trajectory similarity called **TrjSR** (Trajectory via single image Super Resolution), to tackle the important issue of low-quality trajectory data caused by non-uniform sampling rates and noisy data. Specifically, we develop a lighter SISR model to efficiently generate high-quality trajectory images from their low-quality counterparts, and a similarity-aware perceptual loss function is designed, which can improve the performance of similarity computation.

The main contributions of this paper are three-fold:

- We propose a generative model based on SISR to generate a high-quality trajectory image. The inaccurate result due to the non-uniform sampling rates and data noises is alleviated by complementing the incomplete trajectory with the generated trajectory points. To the best of our knowledge, this is the first work of introducing SISR techniques in trajectory similarity computation.
- We design a similarity-aware perceptual loss function to better assess the requirement for trajectory similarity computation. This optimization objective can make our model take the final similarity computation result into account while generating the high-quality trajectory images. By doing so, not only the perceptually better SR result can be obtained, but also the recovered trajectory images can be further embedded into the vector space for measuring similarity.
- With extensive experimental results and evaluation using real-world dataset, we demonstrate that TrjSR achieves

approximately 20% more accurate against the non-uniform sampling rates and 68% more accurate against noises than the state-of-the-art trajectory similarity measurement methods. Besides, we further conduct experiments to study the effectiveness of our model design, and the results prove that we benefit from both accuracy and efficiency.

In the rest of the paper, we review related work in Section II. Section III introduces the problem definition and preliminaries. The technical details of the proposed TrjSR are presented in Section IV, and experimental observations are shown in Section V. Finally, we summarize the paper and outline the future work in Section VI.

II. Related Work

In this section, we briefly review the works related to SISR and trajectory similarity computation.

A. Single image super-resolution

The SISR has been well studied within computer vision research community. There are mainly three types of SISR methods: interpolation based, reconstruction based, and learning based methods.

Interpolation based methods, such as bilinear interpolation [7], bicubic interpolation [8] and Lanczos resampling [9], are extremely straightforward by estimating the HR pixels using their neighborhoods. However, such methods tend to generate overly smooth images and often do not satisfy the reconstruction constraint. Reconstruction based SR methods establish an observation model for the image generation process, and the restored HR image should reproduce the observed LR images after applying the inverse problem of the observation model [10]–[13]. This type of methods usually requires explicit prior knowledge to restrict the possible solution space with the advantage of generating flexible and sharp details. Nevertheless, the quality of the reconstructed image degrades rapidly when the desired magnification factor is large or the number of available input images is small. In these cases, the result may be overly smooth, lacking important high-frequency details [14].

Learning based methods aim to establish a mapping relationship between HR examples (usually from a specific category like human faces or fingerprints) and their LR counterparts through training. Unlike reconstruction based methods using artificially defined prior knowledge, here the prior term for reconstruction is obtained via learning. Note that most of the recent learning based methods fall into the example based methods [15], [16], where the prior knowledge is learned from LR and HR training pairs, thus alleviating the ill-posed nature of SR problems.

Recently, the deep learning based SISR approaches are brought to the attention due to their dramatic superiority to reconstruction based and other shallow learning based methods. Dong et al. [17] first applied convolutional neural network (CNN) to the SR problem and proposed SRCNN, which is the first deep learning based SR method. Since then, various CNN architectures have been studied. To name a few, FSRCNN [18] used a normal deconvolution layer to reconstruct HR images from LR feature maps. The authors of ESPCN [19] proposed an efficient subpixel convolution layer to circumvent the problem of redundant unsampled pixels. Besides, many works have brought about improvements by increasing the network depth, such as VDSR [20], IRCNN [21] and DRCN [22]. Moreover, with the great performance achieved by the ResNet architecture [23], Ledig et al. proposed SRResNet [24], which employs a deep residual network with skip-connection. However, Lee et al. [25] argued that applying original ResNet architecture to low-level vision problems like super-resolution can be suboptimal, so they proposed EDSR to optimize SRResNet by analyzing and removing unnecessary modules to simplify the network architecture, which has achieved relatively better result.

TABLE I: Comparison of some representative deep learning models.

Models	#Depth	#Filters	#Parameters	#Operations
SRCNN	3	64	57K	52.5G
ESPCN	3	64	20K	1.43G
VDSR	20	64	665K	612.6G
DRCN	20	256	1.77M	17974.3G
SRResNet	33	64	1.5M	127.8G
EDSR	65	256	43M	2890G

Note that recent deep learning models like SRResNet and EDSR often have massive parameters, which can cause time and memory issues during the training and inference stages. Besides, a lot of training data are needed to fit the models. A comparison of some of these models is shown in Table I. In our work, we focus on trajectory images instead of photo-realistic RGB images. Since trajectory images have much fewer texture details than the realistic RGB images, we can develop a lighter SISR model to better cope with the time and memory issues.

B. Trajectory similarity computation

Similarity computation has been a fundamental task in trajectory data mining field. There have been a surge of approaches about distance/similarity measurement that can effectively determine the similarity between trajectories. Euclidean distance (ED), also known as L_2-norm, is a parameter-free and linear complexity distance measure, which was originally proposed for time series data since 1960s [26], [27]. Later, ED was extended to measure trajectory similarity [28]–[30] due to the similar data structure between trajectory and time series data. Yet, ED falls short in measuring trajectories with different lengths and suffers from the local time shifting. Dynamic Time Warping (DTW) was proposed to solve the above problems by recursively searching all possible points combination among trajectories with the minimal distance [3]. Piecewise Dynamic Time Warping (PDTW) [31] is another dynamic time warping based similarity trajectory measure, which was improved from DTW. Edit distance with Real Penalty (ERP) [32] is a trajectory similarity computation method based on edit distance, which utilized L_1-norm as the distance measure to seek the minimum number of edit operations required to change one trajectory to another. Longest common subsequence (LCSS)

TABLE II: Similarity Metrics.

Metrics / Methods	Noise	Non-uniform sampling rates	Threshold free	Sequence order free
DTW			✓	
ERP				
LCSS	✓			
EDR	✓			
EDwP	✓	✓	✓	
t2vec	✓	✓	✓	
TrjSR	✓	✓	✓	✓

[4] is a well known measurement. The basic idea of which is to match two sequences by allowing them to stretch, without rearranging the sequence of the elements but allowing some elements to be unmatched. LCSS introduced a threshold ϵ to determine whether or not two points should match, which allows LCSS more robust to noises and outliers. The authors of Edit Distance on Real sequence (EDR) [5] argued that ED, DTW, and ERP are all sensitive to trajectory noises, and LCSS is too coarse to differentiate trajectories with similar common subsequences but different sizes of gaps in between. To solve these problems, EDR also used a matching threshold to alleviate the noise problem like LCSS, and besides, assigned penalties to the gaps between the two matched sub-trajectories according to the lengths of the gaps. Edit Distance with Projections (EDwP) [33] was proposed to mainly cope with the challenge of non-uniform sampling rates. Moreover, EDwP was a threshold-free trajectory similarity measure, which is a relief because usually it is not straightforward to determine the matching threshold. All the aforementioned methods can be classified into pairwise point-matching based methods.

Deep learning models have recently drawn significant attentions, and the ability to learn over large data endows them with more potential and vitality. Li et al. [1] proposed an RNN based model to learn a vector representation of trajectory and compute trajectory similarity between the representation vectors. The result outperforms the conventional similarity measure techniques. Based on [1], Zhang et al. [34] further fused the spatial-temporal characteristics with semantic information and examined the performance on location-based social network datasets. Our work differs from the above RNN based methods in that we utilize SISR model, which is mainly composed of CNN. Table II shows a brief comparison of the aforementioned methods.

III. Problem Formulation

A. Definitions

Definition 1: **Route** is the actual movement of an object in the spatial domain. It is a continuous function from time to space.

Yet, the route cannot be recorded continuously due to the sampling nature of location acquisition techniques.

Definition 2: **Trajectory** is a sequence of spatial-temporal positions sampled from the route and it can be formulated as $T = \{(l_0, t_0), \dots (l_k, t_k)\}$, where l_i is the geometric coordinates and t_i denotes the timestamp at which the moving object passes the location.

In this work, we assume that trajectories share similar route are regarded as similar.

B. Problem description

Given a set of raw trajectories, our goal is to generate the high-quality trajectory images with more details via SISR, and such trajectory images are further embedded into low dimensional vectors for trajectory similarity computation. Our method is expected to be capable of alleviating the low-quality problems caused by non-uniform sampling rates and noises.

IV. The Proposed Method

In this section, we first explain our motivations of using generative model and image representation. Then we describe the generation process of the trajectory image. Next, we introduce how we create example pairs for SISR. Finally, we present our SISR based method to compute trajectory similarity.

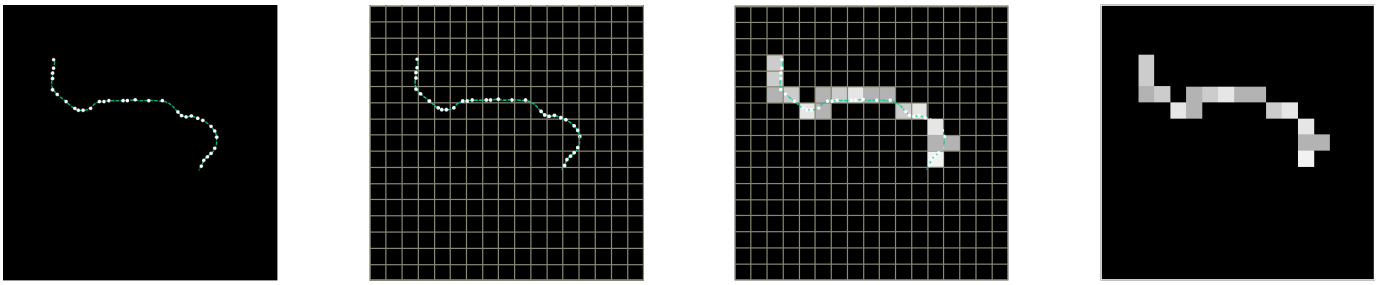
A. Motivations

The main reason that the methods based on pairwise point-matching are unable to handle varying sampling rates is that those methods measure similarity upon trajectories instead of routes. In principle, the route is a better description of the movement since they characterize the entire trajectory shape. However, the route only exists theoretically but not available in practice. Therefore, an intuitive way to tackle non-uniform sampling rates is to infer the underlying route from the raw trajectory data. To this end, we take advantage of the generative model. Specifically, the basic idea is to generate high-quality trajectory images from their low-quality counterparts, where the low-quality images represent the inconsistently sampled trajectories. Those sparse or missing sampling segments will be complemented through the generating process.

The motivation of utilizing the trajectory image to represent raw trajectory is based on the following three observations:

1) Despite raw trajectory data contain spatial information, the sequential form of trajectories does not explicitly reflect the spatial pattern. For example, spatially close positions may not sequentially adjacent to each other;
2) Applying RNN to trajectory sequences suffers from the problem of *exposure bias* and the prediction process cannot take references from the information of the near future.
3) The sequence order is essential when utilizing trajectory sequence to compute similarity, but it is not easy to identify inversely-ordered similar trajectories as we discussed in Section I.

The above observations motivate us to use image representation to tackle these problems. First, the image is naturally spatial-aware, both spatial close and temporal close positions can be depicted as neighboring pixels. Second, using trajectory images, we can take the advantage of CNN to address the problems caused by RNN. For each convolution operation, the



(a) Map coordinates onto region images.

(b) Partition the images into grids.

(c) Assign value to pixels in grid.

(d) The grayscale trajectory images.

Fig. 2: The process of generating trajectory image.

convolutional kernel only actually takes effect around those image patches with non-zero pixels, which avoid predicting trajectory points that are far from the current patch. In addition, the receptive field covered by convolutional kernels contains both former and latter part of the trajectory segments, which means the prediction is not only based on the preceding data but also takes the succeeding data into consider. Third, the representation of trajectory images is not constrained by the sequence order, i.e., trajectories of different orders can be treated as the same trajectory as long as they are sampled from the similar routes.

B. Trajectory image generation

we pre-select a rectangular region that contains sufficient number of trajectories and then scale this region into a fixed-size image. The task ahead is to map the trajectory coordinates collected in this region into the pixels' value of the image.

The overview of the trajectory image generation process is illustrated in Fig. 2. First, we partition the image into grids of equal size, and each grid is comprised of a bunch of pixels. This manner can alleviate the sparsity of the trajectory data, since sparse trajectory points will become spatially closer after being represented by the girds. Then, we assign value to pixels in each grid according to the number of the trajectory points fallen on the gird: the more points imply the longer duration that the object stays in this grid (a rectangular area), and the higher value is assigned to pixels in this grid patch. Thus, different pixel values can be used to capture the temporal property of the trajectory. Finally, we transform the raw trajectory data into a single-channel grayscale image.

C. Creating example pairs

Recall that our method attempts to complement the sparse or missing segments by generating a high-quality trajectory image from its low-quality counterpart. To this end, it is essential to create reliable HR and LR instance pairs. In theory, the trajectory image generated from the underlying route R should be worthy of the HR image since it illustrates the most detailed movement of the moving object. Yet, underlying route is not available in practice, we adopt the relatively high sampling rate trajectory T_{high} and the relatively low sampling rate trajectory T_{low} to create HR and LR image pairs.

(a) LR trajectory image

(b) HR trajectory image

Fig. 3: An example of an LR and HR image pair.

Specifically, given a raw trajectory sequence T^k_{raw}, where k denotes the number of the sampled points this trajectory includes. We assume $T^k_{high} = T^k_{raw}$ as the relatively high sampling rate trajectory. Then we downsample the T^k_{high} by randomly dropping trajectory points to obtain the relatively low sampling rate trajectory $T^{k'}_{low}$, where $k' \leq k$. Both T^k_{high} and $T^{k'}_{low}$ are sampled from the same underlying route R, with only one difference: T^k_{high} is relatively closer to R since it contains more sampled trajectory points.

Furthermore, to create trajectory with the noisy data, we artificially add noises to the trajectory points of $T^{k'}_{low}$. For trajectory point $l(x, y) \in T^{k'}_{low}$, we randomly add Gaussian noise with a radius 100 (meters) to the coordinates:

$$x = x + 100 \cdot d_x, \quad d_x \sim \mathcal{N}(0, 1)$$
$$y = y + 100 \cdot d_y, \quad d_y \sim \mathcal{N}(0, 1)$$

Given the image size, we first transform $T^{k'}_{low}$ into LR image $I^{LR}_{W \cdot H} = G(T^{k'}_{low})$, where W and H are the width and height of the image, and G denotes the trajectory image generation process. To get the HR image, we take $2\times$ as upscaling factor r: $I^{HR}_{rW \cdot rH} = G(T^k_{high})$.

As Fig. 3 shows, the LR trajectory image depicts a trajectory suffering from non-uniform sampling rates and noises. By maximizing the probability $\mathbb{P}(I^{HR}_{rW \cdot rH} | I^{LR}_{W \cdot H})$, we endow the model with the ability to complement the sparse and inaccurate segments from the low-quality trajectory data.

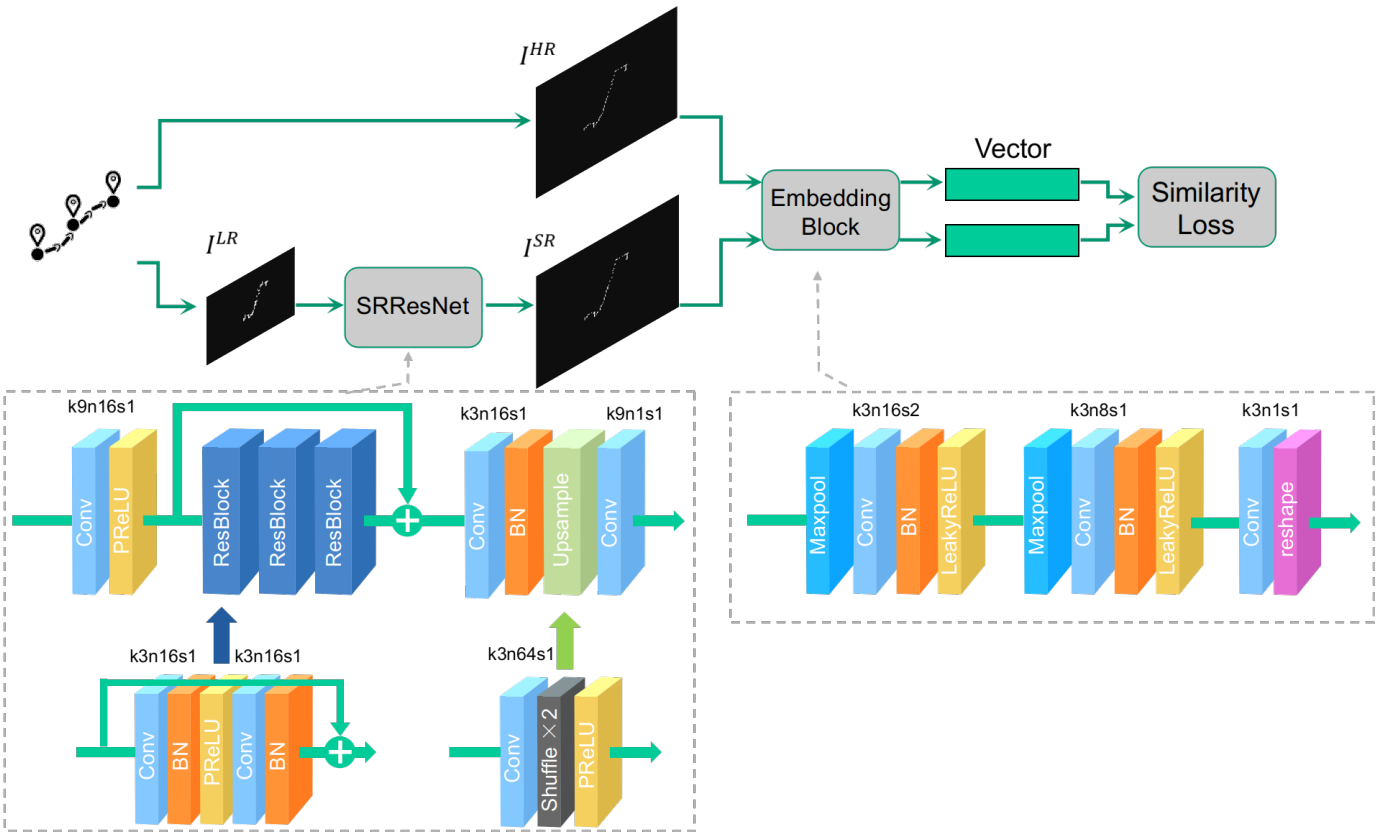


Fig. 4: The architecture of TrjSR with the corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

D. The TrjSR Architecture

The architecture is shown in Fig. 4, which can be mainly divided into three parts: the SISR, the image embedding and the loss function.

1) SISR: To implement SISR, we adopt SRResNet as the base model because it successfully solves the time and memory issues with good performance. Although the authors of EDSR argued that it is suboptimal to directly apply ResNet to the image super-resolution problem, we empirically found that SRResNet performs better in our case based on a range of experiments.

However, the original SRResNet was designed to reconstruct photo-realistic RGB images, which contains 5 residual blocks with 64 feature maps in each convolutional layer. Since our trajectory images are single-channel grayscale images without much realistic texture details, it would be computational expensive to directly apply the original SRResNet to our case. Therefore, to better cope with the time and memory issues, we instead use only 3 residual blocks with 16 feature maps in each convolutional layer. The details of the modified SRResNet are depicted in Fig. 4.

2) Image embedding: The output of SRResNet, i.e. I^{SR}, is not adequate to be directly used for trajectory similarity computation, mainly in two aspects. First, the super-resolved image contains thousands of pixels (in our case, the pixel amount is 256×324), which is time-consuming to compare images pixel-wise. Second, as Fig. 3 depicts, the trajectory points are mapped into non-zero pixels of the image, while most pixels of the trajectory images have zero value. It will be redundant during computation with so many zero-value pixels. Therefore, to efficiently compute similarity among trajectories, we further embed the I^{SR} and I^{HR} into low dimensional vector space as illustrated in Fig. 4. After max pooling and strided convolution layers are used to reduce the image resolution, we stretch the final image (32×40) into a one-dimensional vector. As thus, not only a concise representation of trajectory can be obtained, but also the accuracy and efficiency of similarity computation is much improved, which is demonstrated in Section V-D.

3) Loss function: To train our SRResNet and embed block jointly, we need to design two loss functions to separately evaluate the trajectory similarity and the performance of SISR.

a) Similarity loss: We define our trajectory similarity loss as the Euclidean distance between the embedding vectors from the super-resolved image I^{SR} and the target image I^{HR}.

$$l_{vec} = \frac{1}{|\mathbf{v}|}\|\mathbf{v}^{SR} - \mathbf{v}^{HR}\|_1$$

where $\mathbf{v}^{SR}$ and $\mathbf{v}^{HR}$ denote the embedding vectors of I^{SR} and I^{HR} respectively, and $|\mathbf{v}|$ denotes the dimension of $\mathbf{v}$.

The reason of using L1 distance instead of L2 distance here is that we expect the divergence at each dimension to be wide. However, the L2 distance tends to flatten those divergences. For example, $[0.5, 0.5, 0.5, 0.5]$ and $[1, 1, 0, 0]$ have the same L1 distance value, while the L2 distance gives a smaller loss to the former one. In fact, we prefer the latter one because the big disagreements at the first two dimensions of $[1, 1, 0, 0]$ provide evidence of the greater ability to distinguish vectors. Our experimental result in Section V-E proves that using L1 distance is indeed better than L2 distance in practice.

Using the similarity loss for training our embed block encourages similar trajectory images are as close as possible after being embedded into the vector space.

b) Perceptual loss: The pixel-wise MSE loss is the most widely used optimization target for SISR:

$$l_{MSE} = \frac{1}{r^2WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} \left(I_{x,y}^{HR} - G_{\theta_G} \left(I^{LR} \right)_{x,y} \right)^2$$

In our case, we not only desire the super-resolved image I^{SR} is perceptually similar to the target image I^{HR}, but also expect the I^{SR} can be embedded into a better vector representation for similarity computation. Thus, we design our similarity-aware perceptual loss function as weighted sum of both pixel-wise MSE loss l_{MSE} and similarity loss l_{vec}:

$$l_{SR} = \lambda \cdot l_{MSE} + \mu \cdot l_{vec}$$

where λ and μ denote the weights for the two components. The optimization procedure is formally presented in Algorithm 1.

Algorithm 1 Mini-batch training of TrjSR.

Input: Trajectory image pairs.
Output: The learned SISR model G_{θ_g} and embed block E_{θ_e}.

1: **for** number of training iterations **do**
2: Sample a minibatch of n image pairs: $\langle I_{(i)}^{LR}, I_{(i)}^{HR} \rangle_{i=1}^{n}$
3: Generate the super-resolved images: $I_{(i)}^{SR} = G_{\theta_g}(I_{(i)}^{LR})$
4: Embed the images into vectors:

$$\mathbf{v}_{(i)}^{SR},\ \mathbf{v}_{(i)}^{HR} = E_{\theta_e}(I_{(i)}^{SR}),\ E_{\theta_e}(I_{(i)}^{HR})$$

5: Update the embed block by descending its stochastic gradient:

$$\nabla_{\theta_e} \frac{1}{n} \sum_{i=1}^{n} l_{vec}^{(i)}$$

6: Update SRResNet by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{n} \sum_{i=1}^{n} l_{SR}^{(i)}$$

7: **end for**

V. Experimental Results and Analysis

We evaluate the performance of TrjSR using the real world taxi dataset. Three sets of experiments are performed to analyze the effectiveness of the proposed TrjSR in Section V-B. Examples of our recovered trajectory images from the low-quality counterparts are shown in Section V-C. Besides, we also study the impact of the image embedding and the distance measurement in Section V-D and Section V-E, respectively. Source code, datasets and implementation details are available online at https://github.com/C-Harlin/trjsr.

Fig. 5: The selected area of the main city.

TABLE III: Dataset statistics.

Dataset	#Trips	#Trips in area	#Trips length≥60	#Trips in area length≥60
Porto	1,704,759	1,567,774	416,203	319,384

A. Experiments Settings

1) Dataset: We conduct our experiments on publicly available taxi dataset, in which 1.7M trajectories are collected by 442 taxis running in the city of Porto, in Portugal over 1 year. Firstly, we select a rectangular area in the main city as shown in Fig. 5, where 1.5M trajectories are collected in this area, accounting for nearly 92% of the total trajectory data. Then, we remove trajectories with length less than 60 location points, which yields 319K trajectories. The details are shown in Table III. We use a subset of 200K trajectories to create our training dataset, 20% of which are used for validation.

To create example pairs as described in Section IV-C, we take [1] for reference. Specifically, for a raw trajectory, two types of data transformation methods are applied: downsample and distortion. First, we downsample the trajectory sequence by randomly dropping location points at four different probabilities $[0, 0.2, 0.4, 0.6]$. Then, we distort the above downsampled instances by randomly adding Gaussian noise to coordinates at four different probabilities $[0, 0.2, 0.4, 0.6]$. After the above operations, $4 \times 4 = 16$ trajectories reflecting varying levels of low-quality are created, and they are further transformed into LR images as Section IV-B described. The original raw trajectory is used to generate the HR image. As such, we obtain 16 example pairs $\langle I_{(i)}^{LR}, I^{HR} \rangle_{i=0}^{16}$ from one raw trajectory data.

2) Baseline methods: We compare the accuracy of TrjSR with LCSS [4], EDR [5], EDwP [33] and t2vec [1]. Among

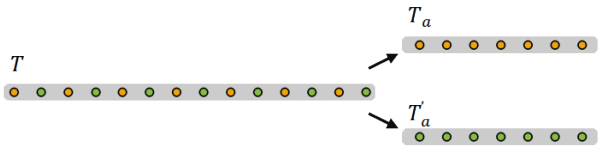


Fig. 6: Illustration of alternately sample trajectory.

them, LCSS, EDR and EDwP are based on pairwise point-matching, and they all have shown superiority over DTW and ERP. t2vec is based on RNN which achieved competitive result in trajectory similarity computation.

3) Training details and parameter settings: Our implementation uses Pytorch [35] on Ubuntu 18.04; training takes roughly 20 hours on a single GeForce RTX 2080Ti GPU. We train our model with Adam optimizer by setting $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, with the learning rate of 10^{-4} and the batch size of 32 for 320,000 iterations, giving roughly 4 epochs over the training data.

For training SRResNet, we use the single-channel grayscale image of size 128×162 as input LR image, with the corresponding HR image of size 256×324 as the target. The weights for similarity-aware perceptual loss function l_{SR} are set as $\lambda = 0.1$, $\mu = 1$. For training the embed block, the input SR/HR images are embedded into the vectors of 1280 dimensions.

We adopt the Java codes provided by the authors of EDwP to implement pairwise point-matching baseline methods. The matching threshold ϵ for LCSS and EDR is set following the original papers [4], [5] described respectively.

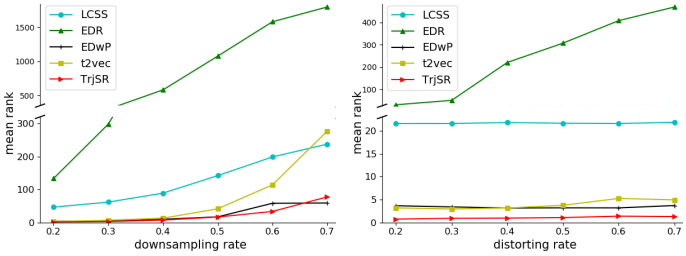
B. Performance Evaluation

Recall that we initially attempt to infer the underlying route R from the raw trajectory T^k_{raw} via a generative model, and those trajectories sharing the similar underlying route are identified as similar. Since R is not available in practice, we alternatively take T^k_{raw} as the suboptimal choice to represent R because it is the closest trajectory to R in dataset. Thus, those low-quality trajectories transformed (i.e. downsample and distortion) from the same T^k_{raw} can be regarded as similar since they all reflect the same pseudo underlying route T^k_{raw}.

With the above idea in mind, we apply the similar procedure as mentioned in [1] by creating two datasets: D_Q and D_B. D_Q contain 1000 trajectories used as the query, and trajectories in D_B are served as database. The size of D_B is a parameter to be evaluated. For each trajectory $T \in D_Q$, we take alternately sample trajectory points to obtain two sub-trajectories T_a and T'_a as shown in Fig. 6. This procedure guarantees that T_a and T'_a are both similar and distinct as well. T_a and T'_a are further used to create two datasets $D_a = \{T_a\}$ and $D'_a = \{T'_a\}$. For each trajectory $T_a \in D_a$, there exists a corresponding similar trajectory T'_a in D'_a. We conduct the same procedure to trajectories in D_B to get D_b and D'_b. Then, for each trajectory $T_a \in D_a$, we retrieve the top-k most similar trajectories in $D'_a \cup D'_b$ (or $D'_a \cup D_b$) and calculate the rank of T'_a. According to our assumption, T'_a should be ranked at the top place since T_a and T'_a are generated from the same trajectory. Besides, we can also perform data transformation (downsample and distortion) on D_a, D'_a and D'_b to evaluate the performance under different types of low-quality.

TABLE IV: Mean rank of 1000 query trajectories versus different database size.

	20k	40k	60k	80k	100k
LCSS	4.429	8.773	12.767	17.178	21.672
EDR	2.213	4.36	6.65	8.53	10.278
EDwP	1.009	1.813	2.564	3.274	3.77
t2vec	0.292	0.5	0.7	0.892	1.046
TrjSR	**0.179**	**0.325**	**0.472**	**0.591**	**0.702**



(a) (b)

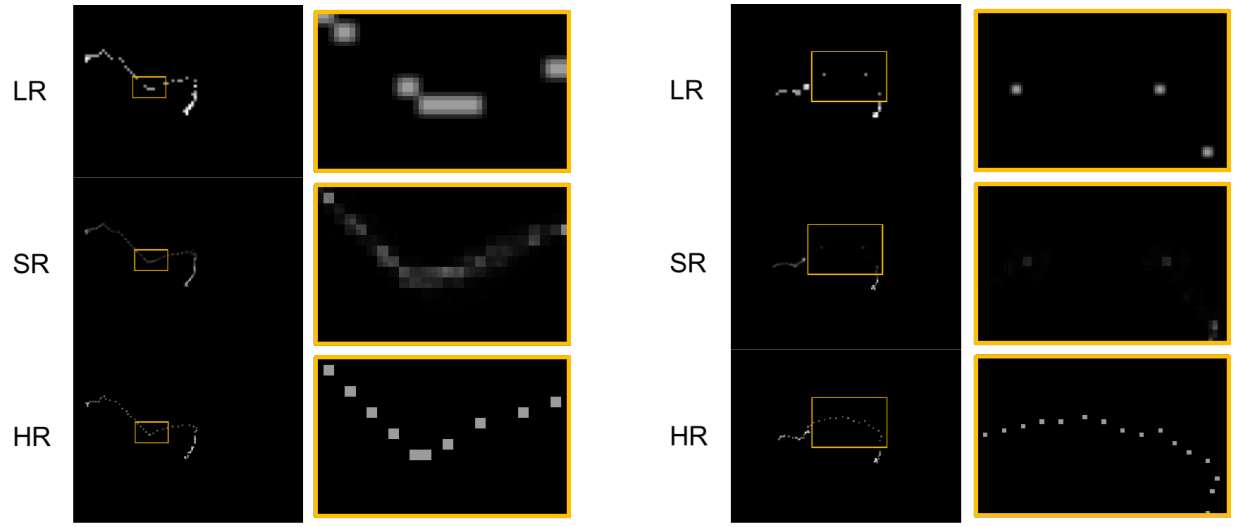
Fig. 7: Mean ranks vary with (a) downsampling rates and (b) distorting rates .

*1) **Accuracy on clean data**:* We first evaluate the performance on clean data, in which the methods are expected to find $T'_a \in D'_a \cup D'_b$ as the most similar trajectory for each $T_a \in D_a$. We increase the size of $D'_a \cup D'_b$ by varying the size of D_B. Euclidean distance is adopted to measure the similarity between trajectories.

Table IV shows the results of the mean ranks of 1000 query trajectories versus the increased size of $D'_a \cup D'_b$ from 20K to 100K. We can see that LCSS performs worst as it ignores the differences of the gap between two similar trajectories, which leads to inaccuracy. EDR achieves slightly better result than LCSS because it assigns penalties to the unmatched point pairs which improves the accuracy. EDwP performs best among all pairwise point-matching methods owing to the technique of linear interpolation it utilizes. Although t2vec demonstrates competitive results, our method outperforms all the other methods, which sheds light on the feasibility of using CNN to deal with sequential trajectory data.

*2) **Accuracy against non-uniform sampling rates**:* Next, we conduct data transformations on trajectories to evaluate the ability against non-uniform sampling rates. Specifically, we randomly downsample trajectory points at the rates varying between 0.2 and 0.7 to simulate the condition of having non-uniform sampling rates. The size of $D'_a \cup D'_b$ is fixed to 100K.

Table V presents the mean ranks of 1000 query trajectories versus different downsampling rates r_1. As the downsampling rate increases, the performance of all methods gets worse, and EDR exhibits the especially worst performance as Fig. 7a



(a) Example of low downsampling rate

(b) Example of high downsampling rate

Fig. 8: Illustration of the effect of downsampling rate on TrjSR.

TABLE V: Mean rank of 1000 query trajectories versus different downsampling rates r_1.

r_1	0.2	0.3	0.4	0.5	0.6	0.7
LCSS	46.777	61.882	89.142	142.188	198.692	237.412
EDR	133.921	298.461	581.49	1076.72	1580.6	1796.89
EDwP	3.42	4.512	9.893	17.214	58.37	**59.132**
t2vec	3.318	6.966	13.211	41.645	114.183	276.632
TrjSR	**1.816**	**3.009**	**7.482**	**16.703**	**33.597**	77.481

TABLE VI: Mean rank of 1000 query trajectories versus different distort rates r_2.

r_2	0.2	0.3	0.4	0.5	0.6	0.7
LCSS	21.596	21.601	21.813	21.66	21.599	21.867
EDR	29.583	49.528	220.296	307.255	408.663	470.05
EDwP	3.663	3.41	3.166	3.204	3.198	3.706
t2vec	3.195	2.952	3.177	3.764	5.26	4.941
TrjSR	**0.745**	**0.923**	**0.971**	**1.081**	**1.392**	**1.305**

depicts. t2vec performs relatively well when r_1 is low, but its mean ranks degrade quickly as r_1 increases, which implies that t2vec is not competent enough against non-uniform sampling rates. Note that the result of t2vec in our case is different with the experimental result presented in the original paper [1]. This probably results from the fact that t2vec partitions a larger area into cells to learn distributed cell representations, therefore the cell size could be too coarse to fit a relatively small area like the main city in our experiment.

TrjSR performs best when r_1 is in the range of 0.2 to 0.6, and achieves approximately 20% more accurate than EDwP on average. To understand why EDwP is a little bit better than TrjSR when r_1 at 0.7, we check the super-resolved trajectory images. Fig. 8a exhibits an example of SISR when r_1 is low, as we can see, our model tries to interpolate the missing part between sequential points. However, when r_1 is extremely high, as Fig. 8b shows, TrjSR can hardly reconstruct trajectory segments anymore due to the lack of necessary anchor points. On the contrary, EDwP does insertion based on the assumption of linear interpolation. It can handle such a worst situation by linear interpolating anyway. Since such an extreme case is rare in practice, we can conclude that our method is competent against non-uniform sampling rates in most situations.

*3) **Accuracy against noises**:* Then, we study the effect of noises on different methods. We randomly distort the coordinates of trajectory in D_a and $D'_a \cup D'_b$ to simulate the situation where noises exist. The distorting rate r_2 varies from 0.2 to 0.7 and $|D'_a \cup D'_b| = 100,000$.

From Fig. 7b, we can intuitively observe that the performance of all methods, except for EDR, does not degrade much as r_2 increases, which means they are all robust to noises to some extent. Similar to the case of downsample, the mean rank of EDR starts to degrade dramatically when r_2 increases. EDwP and t2vec demonstrate similar results. TrjSR outperforms all the other methods with a large margin when r_2 varies between 0.2 and 0.7, which averagely yields an improvement of 68% in accuracy compared to EDwP, indicating that our method is the most robust against noises when computing trajectory similarity.

C. Result of SISR

Recall that we aim to generate high-quality trajectory images from low-quality ones through SISR, here we present the qualitative results of our trajectory SISR as shown in Fig. 9. The proposed model successfully complements the incomplete segments with generated points and also reconstructs the details from noises.

D. Effect of image embedding

As we discussed in Section IV-D2, a trajectory image is not adequate to be directly used for similarity computation. Therefore, we embed trajectory images into a vector space. In

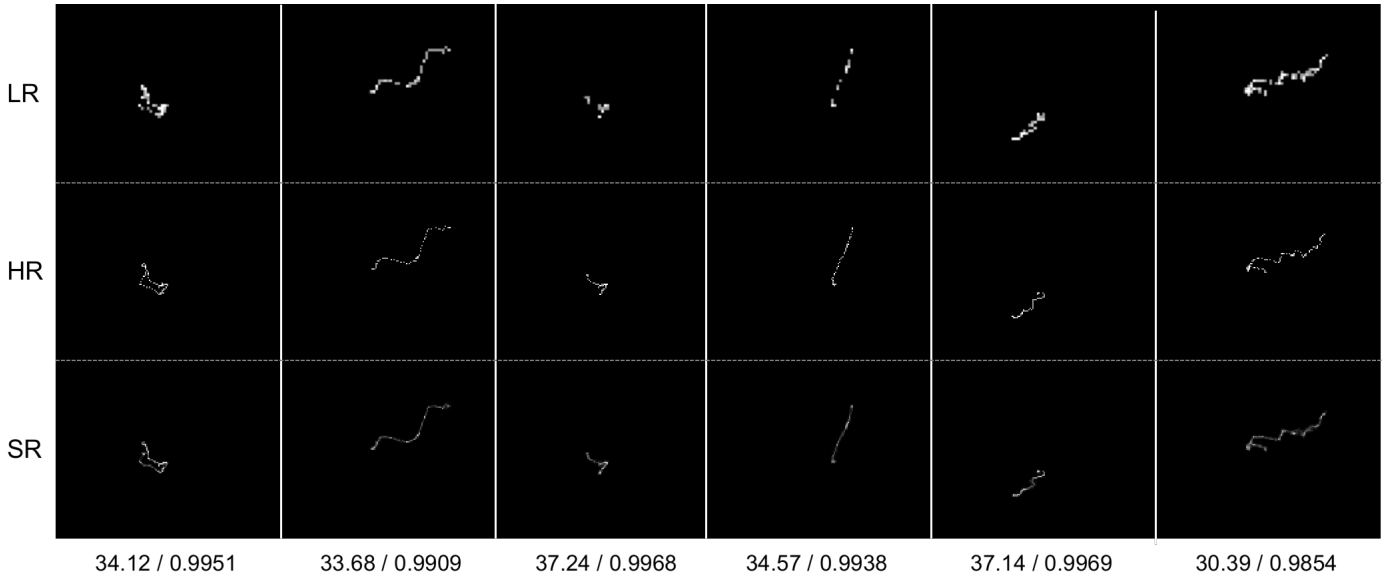


Fig. 9: Qualitative results (PSNR(dB) / SSIM) of trajectory image $\times 2$ super-resolution.

TABLE VII: Mean rank for different similarity measurements versus different database sizes.

	1k	2k	3k	4k
MSE	0.015	0.032	0.044	0.063
SSIM	0.082	0.187	0.257	0.341
Embedding	**0.01**	**0.017**	**0.028**	**0.041**

this section, we evaluate the effectiveness of image embedding for trajectory similarity computation.

To measure the similarity of trajectory images directly for comparison, we train our modified SRResNet from scratch as the authors of SRResNet [24] implemented. The training process takes the same parameter settings as we use in the process of training TrjSR. As for the similarity measurements, we utilize MSE and structural similarity index measure (SSIM), which are commonly used similarity loss functions in the field of SISR. Similar to the implementation in Section V-B1, we retrieve the rank of $T'_a \in D'_a \cup D'_b$ for each $T_a \in D_a$. The size of $D'_a \cup D'_b$ is reduced to the range between 2K and 4K, because the time consumption for calculating SSIM on large image datasets is unaffordable.

The results in Table VII prove that simply minimizing MSE or SSIM on trajectory images does not bring about the most accurate result in similarity computation. Instead, with the trajectory image embedding, we can improve the accuracy of similarity computation as well as obtain the concise trajectory representation, which benefit our solution in time consumption when calculating the similarity. Although we first embed trajectory images into vectors before calculating the similarity, the embedding process can be done very fast on GPU. Then, it takes only linear time complexity $O(|\mathrm{v}|)$ to measure the similarity between trajectories. As shown in Fig. 10, using image embedding achieves up to 12 times faster than directly applying MSE on trajectory images.

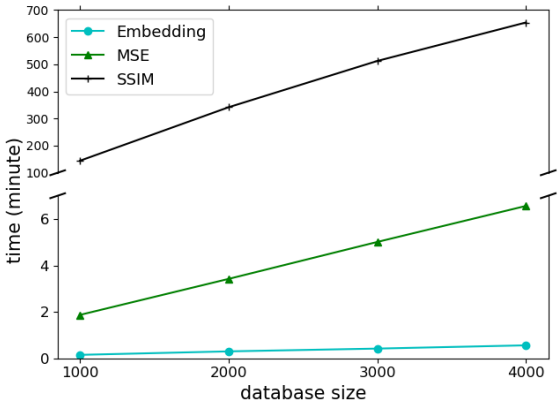


Fig. 10: Query efficiency of different similarity measurements versus different database sizes.

E. Effect of L1 and L2 distance

To prove that L1 distance is better than L2 distance for measuring the similarity loss in our case as we analyzed in Section IV-D3a, we conduct the following experiment to compare their effects on the accuracy of trajectory similarity computation.

We first pre-train our SRResNet independently for 3 epochs to make it generate super-resolved images with good-enough quality. Then, two embed blocks with the only difference of L1 distance and L2 distance in similarity loss function l_{vec} are used to jointly train with SRResNet for another 1 epoch respectively. Similar procedure for evaluating accuracy is applied as we did in Section V-B2 and Section V-B3.

As shown in Table VIII, under the same training condition, the accuracy of using L2 distance declines rapidly as down-sampling rate increases. As for distortion, using L1 distance

TABLE VIII: The impact of different distance measures.

Distance	downsampling rate r_1			distorting rate r_2		
Measure	0.2	0.4	0.6	0.2	0.4	0.6
L2	**1.892**	94.759	142.065	1.246	1.702	12.674
L1	3.137	**12.806**	**85.234**	**0.739**	**1.322**	**1.157**

brings about more robustness against distorting rate. From the above, we can conclude that L1 distance is more adequate to be used in similarity computation in our case. Besides, it is more computational expensive to compute L2 loss compared to L1 loss.

VI. Conclusion

In this paper, we proposed a novel generative model, TrjSR, to deal with the low-quality trajectory data caused by non-uniform sampling rates and noises in trajectory similarity computation. The SISR techniques are adopted to generate high-quality trajectory images, and we developed a similarity-aware perceptual loss function to encourage our model to generate high-quality trajectory images in favor of computing similarity. In this sense, the proposed TrjSR bridged the gap between the sequential trajectory data and the SISR. Extensive experimental results showed that the proposed method outperforms existing methods in terms of accuracy and efficiency. Moreover, we conducted comparative study to further validate the effectiveness of our model design. In addition, the embedding vector of trajectory images can be used not only for similarity computation but also as the trajectory representation for other applications, which we leave for our future investigation.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) (Grant No. 52071312), and the Open Program of Zhejiang Lab (Grant No. 2019KE0AB03).

References

[1] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*, 2018, pp. 617–628.

[2] N. National Coordination Office for Space-Based Positioning and Timing. (2020) How accurate is gps? [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/#how-accurate

[3] B. K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *Proceedings - International Conference on Data Engineering*, 1998.

[4] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," *Proceedings - International Conference on Data Engineering*, 2002.

[5] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 491–502.

[6] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015.

[7] X. Li and M. T. Orchard, "New edge-directed interpolation," *IEEE Transactions on Image Processing*, 2001.

[8] R. G. Keys, "Cubic Convolution Interpolation for Digital Image Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1981.

[9] C. E. Duchon, "LANCZOS FILTERING IN ONE AND TWO DIMENSIONS." *Journal of applied meteorology*, 1979.

[10] M. E. Tipping and C. M. Bishop, "Bayesian image super-resolution," in *Advances in Neural Information Processing Systems*, 2003.

[11] J. Sun, J. Sun, Z. Xu, and H. Y. Shum, "Image super-resolution using gradient profile prior," in *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[12] S. Dai, M. Han, W. Xu, Y. Wu, Y. Gong, and A. K. Katsaggelos, "Softcuts: a soft edge smoothness prior for color image super-resolution," *IEEE transactions on image processing*, vol. 18, no. 5, pp. 969–981, 2009.

[13] Q. Yan, Y. Xu, X. Yang, and T. Q. Nguyen, "Single image superresolution based on gradient profile sharpness," *IEEE Transactions on Image Processing*, 2015.

[14] S. Baker and T. Kanade, "Limits on super-resolution and how to break them," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[15] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *International Journal of Computer Vision*, 2000.

[16] J. Sun, N. N. Zheng, H. Tao, and H. Y. Shum, "Image Hallucination with primal sketch priors," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[17] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[18] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *European conference on computer vision*. Springer, 2016, pp. 391–407.

[19] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[20] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[21] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[22] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[24] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[25] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced Deep Residual Networks for Single Image Super-Resolution," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[26] M. Priestley, "State-dependent models: A general approach to non-linear time series analysis," *Journal of Time Series Analysis*, vol. 1, no. 1, pp. 47–71, 1980.

[27] P. E. Pfeifer and S. J. Deutsch, "A Three-Stage Iterative Procedure for Space-Time Modeling," *Technometrics*, 1980.

[28] F. Clarke, "Optimal solutions to differential inclusions," *Journal of Optimization Theory and Applications*, vol. 19, no. 3, pp. 469–478, 1976.

[29] R. Jonker, G. De Leve, J. Van Der Velde, and A. Volgenant, "Rounding symmetric traveling salesman problems with an asymmetric assignment problem," *Operations Research*, vol. 28, no. 3-part-i, pp. 623–627, 1980.

[30] A. C. Sanderson and A. K. Wong, "Pattern trajectory analysis of nonstationary multivariate data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, no. 7, pp. 384–392, 1980.

[31] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[32] L. CHEN and R. NG, "On The Marriage of Lp-norms and Edit Distance," in *Proceedings 2004 VLDB Conference*, 2004.

[33] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, and S. Raghavan, "Indexing and matching trajectories under inconsistent sampling rates," in *Proceedings - International Conference on Data Engineering*, 2015.
[34] Y. Zhang, A. Liu, G. Liu, Z. Li, and Q. Li, "Deep Representation Learning of Activity Trajectory Similarity Computation," *2019 IEEE International Conference on Web Services (ICWS)*, pp. 312–319, 2019.
[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.