# TrajDistLearn: Learning to Compute Distance between Trajectories

Janit Anjaria, Hong Wei, Hao Li, Shlok Mishra, Hanan Samet
Department of Computer Science
University of Maryland
College Park, Maryland 20742
{janit,hyw,haoli,shlokm,hjs}@cs.umd.edu

## ABSTRACT

Discovering and clustering similar trajectories is a cornerstone task for movement pattern analysis and location prediction in applications like ride-sharing, supply-chain, maps and autonomous driving. However, the existing distance computation is computationally expensive and is hard to parallelize, which makes the large-scale computation prohibitive. We propose TrajDistLearn, a unified learning-based approach for trajectory distance computation, in which the traditional point-based trajectories are converted into rasterized images, and the distance function is learned via Siamese Networks in an end-to-end way. The framework accurately learns various distance metrics for the trajectory similarity computation, including the widely used Fréchet distance, which is a computationally expensive distance metric. The efficiency gain with neural network approximation is significant. Our approach achieves at least a 3000x speed-up on GPU and a 40x speed-up on CPU in comparison with naive Fréchet distance computation. In addition, our approach's computational overhead is independent of the sampling rate of the trajectories. Extensive experiments on real-world trajectory datasets demonstrate the effectiveness and efficiency of TrajDistLearn.

## CCS CONCEPTS

•**Computing methodologies** →*Knowledge representation and reasoning;* Image representations; Neural networks;

## KEYWORDS

deep learning, trajectory, similarity, metric learning, frechet, transportation

**ACM Reference format:**
Janit Anjaria, Hong Wei, Hao Li, Shlok Mishra, Hanan Samet. 2021. TrajDistLearn: Learning to Compute Distance between Trajectories In *Proceedings of 14th ACM SIGSPATIALInternational Workshop on Computational Transportation Science , Beijing, China, November 1, 2021 (IWCTS'21),* 10 pages.

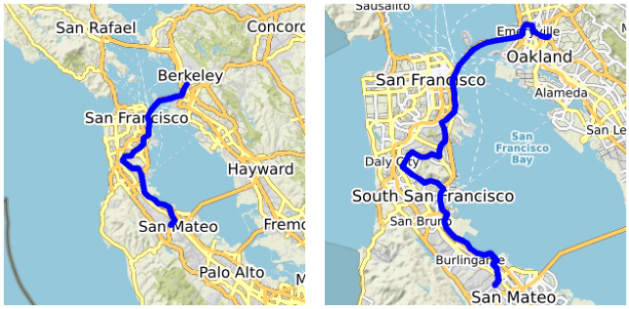



**Figure 1: The two images represent two different location trajectories plotted on OpenStreetMap. We validate that our approach generalizes to multiple distance metrics and various real-world datasets. For Fréchet distance computation, our approach is independent of the number of segments in a trajectory, unlike other Fréchet distance approximation methods. TrajDistLearn approximates the Fréchet distance for the given trajectory pair as 4429.66 units, and the actual distance is 4409.09 units. Therefore, the relative distance is 0.004, and we observe atleast a 3000x speed-up on GPU and atleast a 40x speed-up on CPU compared to Naive Fréchet distance method. [4].**

## 1 INTRODUCTION

Due to the enormous amount of spatial trajectory data collected from GPS-equipped devices and location-based services, efficient matching and querying for trajectories has become an important practical and research topic. The most widely used application is that of a ride-sharing service, where it is required to compute the distance between pairs of trajectories and answering similar-path queries in real-time to optimize carpooling. Additionally, shape matching between geometric objects also involves polygonal curve similarity computation [8]. There are various geometric definitions to measure trajectory similarity [9, 10, 24, 27]. One of the prevalent distance metrics has been Fréchet distance [4] due to its parameterized nature of the trajectory representation. It measures two curves' similarity considering the distances of all discrete points on the two curves and their paths. Note that these are different distances used in road networks e.g. [34–36] or medial axis transforms e.g. [32, 33].

Although Fréchet distance [24] has shown extensive usage across multiple applications, its computational complexity remains a bottleneck. Additionally, current algorithms for approximating Fréchet distance like the Discrete Fréchet distance [21] depend on the number of segments in a trajectory. Hence, the best possible complexity is proportional to the number of points on each of the curves. These pointwise matching techniques for distance computation suffer in


*IWCTS'21, Beijing, China*

DOI: 10.1145/3486629.3490693

scenarios with a non-uniform trajectory sampling rate. There have been recent advances in solving the trajectory similarity computation problem using Recurrent Neural Networks. However, these methods generally modify the trajectory's sampling rate, involve many hyperparameters, which makes training non-trivial and, more importantly, do not solve the distance computation problem [43]. We argue that with the rapid development of alternate sources of mobile power and with the enormous amount of location information for trajectories - a good trajectory distance computation technique should be accurate, scalable to larger datasets and generic for various distance metrics.

It is important to note that the complexity in tasks like trajectory similarity computation or trajectory clustering lies in the trajectory distance computation [4, 21, 26]. We approach this problem by breaking it down into two subproblems: (1) Trajectory representation (2) Trajectory distance computation. To avoid losing trajectory sampling information and creating a robust trajectory representation, we propose to use rasterized images where pixels depict the location, i.e. latitude and longitude, on a spatial grid. Such a trajectory representation enables us to design an agnostic approach to the length of the trajectory and not lose the trajectory sampling information. As for trajectory distance computation, we use a very lightweight neural network with just three convolutional layers and utilize the Siamese networks to map similar trajectories closer in the embedding space.

To summarize, the paper makes the following contributions:

- We propose TrajDistLearn, an end-to-end learning based framework for trajectory distance computation. The distance function between two trajectories is learned and approximated by neural networks rather than hand-designed algorithms. To the best of our knowledge, it is the first deep learning based framework for trajectory distance computation.
- We represent the trajectories as rasterized images, which are robust to the trajectory sampling rate. Using rasterized images for trajectory representation enables our approach to be robust and lead to better speed for distance approximation. For computationally complex distance metrics like Fréchet distance, TrajDistLearn is the first Fréchet distance approximation method independent of the trajectories' sampling rate. TrajDistLearn achieves at least a 3000x speed-up speed-up on GPU and 40x speed-up on CPU compared to naive Fréchet distance computation or other approximation techniques.
- Our framework can generalize to different distance metrics across different types of real-world datasets. By using cosine similarity as a similarity measure, we validate that the features produced by TrajDistLearn can be effectively used for applications like trajectory similarity computation.

## 2 RELATED WORK

*Trajectory Distance Metrics.* There has been substantial work on measuring the similarity among trajectories. For example, the Hausdorff distance [26] emphasizes the distances between vertices of trajectories by measuring how far two subsets of a metric space are from each other. Traditional approaches to computing the Hausdorff distance between two trajectories perform a linear scan over one trajectory. A spatial index like RTree [25] is often utilized to help compute the nearest neighbour in the other trajectory for each vertex in the former trajectory. The best of these measures for trajectory distance computation and similarity is based on dynamic programming techniques to identify the optimal alignment, leading to $O(n^2)$ computation complexity. Therefore, these techniques prove inefficient as similarity measures for clustering in a large database of trajectories. Distance computation remains to be the computational bottleneck for all the different similarity measures. Among these distance definitions, the Fréchet distance is exciting due to its parameterized trajectory representation. Although the Fréchet distance finds use in many applications, such as map matching [3, 11, 39], its computation remains a challenge. The traditional solution for computing the Fréchet distance for a pair of trajectories $P$ and $Q$ with $p$ and $q$ segments has a time complexity of $O(pq\log(pq))$ [4].

*Traditional Fréchet distance methods.* Direct computation of this distance metric between polygonal trajectories is very time-consuming [4, 12]. Some work focuses on solving variants of the problem [14, 16–18]. For example, Cook et al. [23] describe a polynomial-time algorithm to compute the geodesic frechet distance between two polygonal curves in a simple polygon. Chambers et al. [17] describe a polynomial-time algorithm to compute the homotopic Fréchet distance between polygonal curves in the Euclidean plane with obstacles. If we relax the requirement of continuously sweeping every point on the given curves by only examining the vertex positions of polygonal curves, we have Discrete Fréchet distance. Eiter et al. [21] present a polynomial-time solution using dynamic programming for computing the Discrete Fréchet distance. Discrete Fréchet distance is dynamic programming with a $O(pq)$ time complexity. Hence, even this approximation depends on the number of segments forming the trajectory. By using the problem's geometry to encode legal positions of the moving targets on the trajectories as states of finite automation, Agarwal et al. [1] first present an algorithm that runs in subquadratic time in the standard RAM model. Similarly, [6] provides a near-linear time approximation algorithm for the Discrete Fréchet distance by using curve simplification to speedup the algorithm [2].

There has been recent work in optimizing Fréchet distance computation by [7, 13, 20] and has demonstrated substantial improvement in runtime. The common theme for all the algorithms remains to be hand-designed algorithms specifically for Fréchet distance computation. All of these approaches downsample or limit the number of locations while creating the trajectories, are not generic for other distance metrics, and, most importantly, are not data-driven, limiting practical applications.

*Neural Networks based Trajectory Representation.* With the advent of deep learning, there has been recent work in representation learning for trajectories. The work done by Yao et al. [41] is one of the earliest attempts to represent trajectories as recurrent networks. It is used to detect space and time-invariant trajectory clusters by employing a sliding-window-based approach to extract robust movement features and learn fixed-length representations for the trajectories. Li et al. [30] also utilizes a sequence to sequence model along with spatial proximity aware loss function. Recent work by

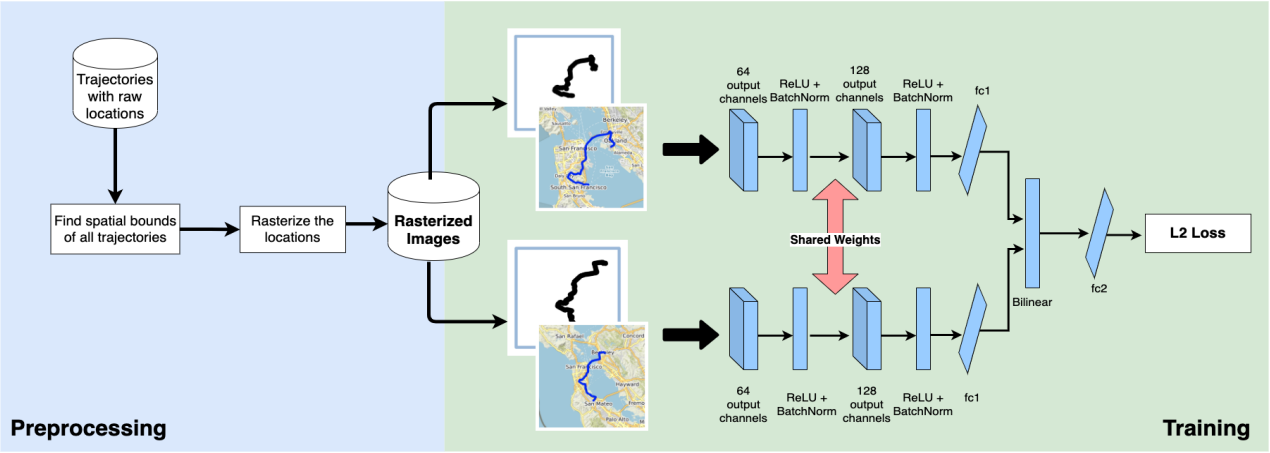



**Figure 2: Overview of the framework. The figure explains the various steps of pre-processing raw trajectory location data to create rasterized images for trajectory representation. The trajectories are represented as raw rasterized images or map images used to train the neural network to learn and output distance among the trajectory pairs.**

Zhang et al. [43] also use recurrent neural networks for trajectory representation, along with a non-trivial cost-function for trajectory similarity computation. This work does not solve the distance computation problem, uses a lot of heuristics and hyperparameters, which makes training non-trivial and most importantly, uses a fixed-length trajectory representation by modifying the sampling rate of the trajectory.

*Trajectory Similarity using Neural Networks.* There has also been some recent work in using Hausdorff distance for object detection without bounding boxes [31]. Additionally, [42] computes Discrete Fréchet distance by replacing dynamic programming with interactions among adjacent cells in convolutions. The proposed model is hard to interpret in terms of their representation of the Discrete Fréchet distance, depends on the number of segments in the trajectory and does not solve the computationally expensive continuous Fréchet distance. The work done by [37] discusses how they use the euclidean distance between features generated by using representations for time series data. Nevertheless, they do not solve the Fréchet distance computation problem, which is a regression problem to learn, and not a classification problem and thereby remains a computational challenge.

However, all of the aforementioned approaches to trajectory representation or distance calculation have multiple shortcomings. The traditional methods for distance calculation are hand-designed and, thereby, lack robustness. The neural network based trajectory representations downsample the trajectories and are harder to train. Therefore, all of the existing approaches are computationally intensive and do not solve the distance computation problem.

## 3 OUR APPROACH

For practical reasons, trajectories are usually represented as a sequence of time-stamped spatial locations

$$T = [(p_1, t_1), (p_2, t_2), ..., (p_m, t_m)] \tag{1}$$

where $p_i$ and $t_i$ represent the spatial location represented as latitude and longitude (or x and y coordinates) at a given time $i$. The assumption is that the linear interpolation between subsequent samples is a sufficiently accurate reconstruction of the movement of the object between two consecutive location samples. In the language of GIS, therefore, a trajectory is represented as a LINESTRING feature. For our approach, we propose to learn the distance function $D(T_1, T_2)$, where $T_1$ and $T_2$ are the two trajectories among which we need to compute distance.

For TrajDistLearn, trajectories are represented using rasterized images where each pixel represents a location (latitude and longitude) within spatial bounds. This sort of a trajectory representation enables us to learn complex distance functions like the Fréchet distance, which is represented as:

$$d_F(T_1, T_2) = \inf_{\alpha,\beta} \sup_{t\in[0,1]} ||T_1(\alpha(t)) - T_2(\beta(t))|| \tag{2}$$

The Fréchet metric takes into account the flow of the two curves because the pairs of points whose distance contributes to the Fréchet distance sweep continuously along their respective curves. For the same reason, Fréchet distance is better for measures like similarity as compared to its counterparts like Hausdorff distance. Further, we represent the trajectories by plotting them on a map and a white background. We thereby view distance measurement among trajectory pairs as computing the difference(or similarity) among images. The goal remains to utilize a network architecture that can map similar trajectories closer in the embedding space than dissimilar trajectories. Siamese networks were introduced initially by [15] to solve signature verification as an image matching problem. Weight sharing in Siamese Networks guarantees that two similar images could not be mapped to very different locations by their respective networks.

### 3.1 Preprocessing

From a formal point of view, trajectories can be represented as a continuous map $T$ (sequence of time-stamped locations) from the unit


(a) Map image (b) Raw rasterized image

**Figure 3: Examples of trajectory representation for a trajectory from the GISCUP dataset. We extract the trajectory representation (b) from the trajectory representation (a). In our pre-processing step, we do not make any changes to the sampling rate or representation of the trajectory.**

interval $[0, 1]$ to space. In this setting, the unit interval represents the time along the trajectory, so given a trajectory $T : [0, 1] \rightarrow R^2$, the time 0 is mapped to the first point $t(0)$ of the trajectory and $t(1)$ points to the last point. We propose TrajDistLearn, the first trajectory distance computation method that uses rasterized images for trajectory representation for distance computation. We create two different types of rasterized images for the trajectories under consideration: (1) Raw Rasterized Images (2) Map Images for the GISCUP Dataset[1]. Examples of both the types of images for a given trajectory are shown in Figure 3. We find the raw rasterized images' dataset's maximum spatial bounds and plot the raw location on an image. For Map Images we project the location data from the $epsg : 3857$ to $epsg : 4326$ coordinate system and plot the corresponding location trajectories on OpenStreet Map[2].

## 3.2 Architecture

We propose using a very lightweight network architecture consisting of two convolutional layers (64 and 128 output channels). Each of these layers is followed by ReLU activations and batch normalization [28]. Followed by the convolutional layers is a fully connected layer for each of the siamese twins, the features from which are fused using a bilinear layer. This resultant feature is further passed through a fully connected result to produce a 1D result. This layer's result is compared against the actual distance [4, 21, 26] using the MSE loss function. Therefore, TrajDistLearn solves the trajectory distance computation problem as a regression problem. Convolutional layers for this task enable us to learn the appropriate subsequences from the input. Using this architecture with a max-pooling layer makes the network translation invariant and robust to noise in the input data. This network proves to be very robust, quick to train and intuitive to understand to the extent where we can use the same network for both datasets and all distance metrics despite of the significant difference in the sparseness of the underlying data for the two datasets.

[1]https://github.com/TWTDIG/GISCUP17TUE
[2]https://www.openstreetmap.org/about

As discussed earlier, we incorporate the actual distance [4, 21, 26] between the trajectory pairs in the loss function to guide similar trajectories closer in the embedding space. It is important to note that the trajectories are purely represented as rasterized images (raw and map images) and do not include direction as a meta-feature for the trajectories in our task, as distance computation between trajectory pairs does not fundamentally require the direction. It is also important to note that the current representation does not explicitly encapsulate the temporal nature of the trajectory. Therefore, with the L2 loss function encapsulating the actual distance, TrajDistLearn can generate an approximation for the distance metric for a given pair of input trajectories.

## 3.3 Learning trajectory similarity

As shown in Figure 2, the features generated by each of the siamese twins being trained with shared weights would represent the similarity relationships among the trajectories. We thereby propose using the most intuitive Cosine Similarity as a metric for similarity for the trajectory features. So, higher cosine similarity would imply a lesser distance between the trajectories and, thereby, higher similarity. We validate with experiments with multiple distance metrics and datasets - that TrajDistLearn is not only useful for distance computation, but it also solves the trajectory similarity computation problem.

# 4 EXPERIMENT

## 4.1 Experimental Setup

*Datasets.* We use two datasets based on the variance in the dataset and the diversity of the type of underlying trajectory data for evaluating our framework. First is the Character Trajectory Dataset [19], which is used to generate 50,000 unique training trajectory pairs and 5000 unique testing trajectory pairs. Second, is the GISCUP dataset[3][40], which is used to generate around 19,000 unique training trajectory pairs and around 5000 unique testing trajectory pairs. The dataset information is in Table 1. Samples of rasterized trajectory images are represented in Figure 3. Additionally, Table 2 represents how despite of the significant difference in the variance of the underlying raw trajectory location data as shown in Table 1, the variance in the rasterized image is not significantly different.

| Dataset | Min | Max | Mean | Std. Dev | Variance |
|---|---|---|---|---|---|
| Character | 60 | 182 | 119.83 | 20.99 | 440.90 |
| GISCUP | 10 | 768 | 247.89 | 154.07 | 23738.30 |

**Table 1: Statistics about the length of trajectories for both the Character Trajectory and GISCUP dataset.**

| Dataset | Variance |
|---|---|
| Character | 1742.46 |
| GISCUP | 1107.12 |

**Table 2: Variance in the rasterized image representation for all the trajectories for both the datasets.**

[3]https://github.com/TWTDIG/GISCUP17TUE

*Evaluation Criterion.* Following quantitative criteria are employed to evaluate the experiments:

- *Accuracy Measurement.* Computing distance between trajectories serves multiple downstream applications. The nature of the trajectories and also the domain of application determines the accuracy of distance computation required. We report accuracy by calculating how far the network computed value is from the actual distance, which is calculated using the pure traditional distance functions [4, 21, 26]. Therefore, to evaluate our accuracy, we introduce the metric of relative distance, which is defined as follows:

$$\text{relative distance} = \frac{|\hat{D} - D_{GT}|}{D_{GT}} \tag{3}$$

  where $\hat{D}$ is the distance computed by our network and $D_{GT}$ is the ground-truth i.e. distance value computed using the distance functions [4, 21, 26] for a given pair of trajectory. We normalize with $D_{GT}$ to ensure we capture the variation in the distance among all trajectory pairs, and also provide a unified evaluation metric.
- *Speed.* As a special case for Fréchet distance, which is difficult to compute both from a computational complexity perspective as well as from an implementation point of view in that it is not easy to get all details right and efficient on real computers, we measure the time complexity (or computational performance) for Fréchet distance computation. We measure the time complexity or speed and performance by looking at the following two further sub-metrics: (1) Dependence of time complexity on the number of segments(or length) of the trajectory (2) Average performance, in terms of speed on a shuffled dataset. Less dependence on the length of the trajectory and higher speed on a shuffled dataset would mean more robustness and reliability for the method.

*Training Settings.* Even with the significant difference in the raw trajectory location data for the two datasets, we are able to use the same hyper-parameters for all distance metrics for both datasets for a similar trajectory representation i.e. raw rasterized images and map images. As an overview, the optimizer used is the Adam Optimizer [29] with learning rate of 1e-8 for raw rasterized images and 0.0001 for map images, and learning schedule being a multi-step learning rate change at $5^{th}$, $10^{th}$, $15^{th}$ and $20^{th}$ epochs and $\gamma$=0.01 for both datasets and all distance metrics.

*Baseline Methods.* We use the traditional pure distance computation methods for our experiments for Fréchet , Discrete Fréchet , and Hausdorff distance computation [4, 21, 26] for accuracy evaluation. We use the distance value obtained from these methods to obtain the relative distance. Additionally, for the speed and performance evaluation, we also compare TrajDistLearn against other Fréchet distance approximation techniques that are widely used or recently been developed, such as Eiter et al. [21] and Dutsch et al. [20]. This is done to compare our neural network based distance learning and approximation approach against the hand-designed algorithms for Fréchet distance computation and approximation.

## 4.2 Results

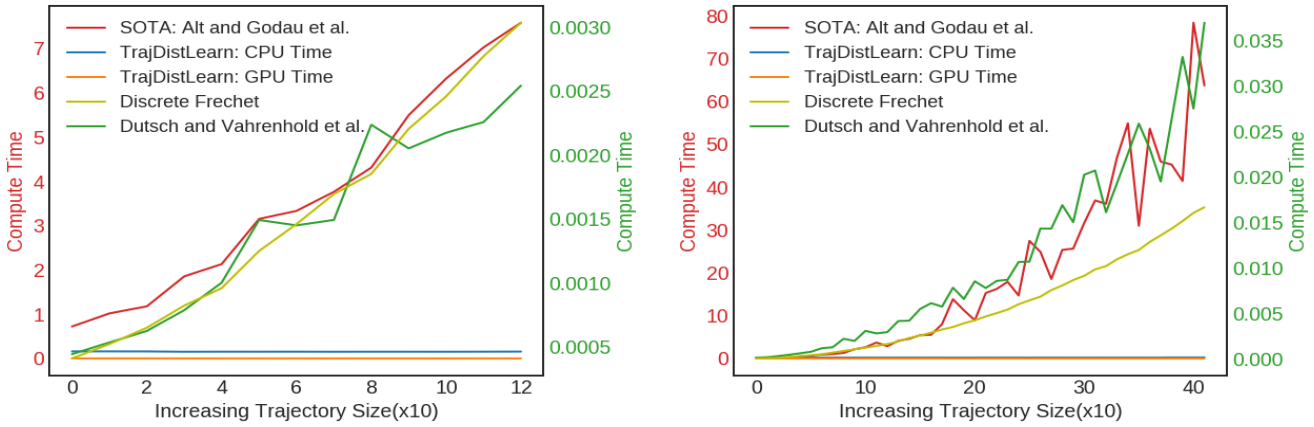
*Accuracy.* We evaluate our approach for the widely used distance metrics, namely Fréchet , Discrete Fréchet and Hausdorff distance for both the datasets. Table 3 articulates the accuracy(relative distance) for both datasets and all distance metrics. It demonstrates that TrajDistLearn has a $relative\ distance \leq 0.5\ unit$ for an average of **97.81%** across all distance metrics for the Character Trajectory dataset, and an average of **100%** across all distance metrics for the GISCUP dataset. Relative Distance serves as a stable metric of accuracy evaluation because it takes into consideration the actual distance which is important to capture variance of computation at test time. Table 4 and Table 5 compare TrajDistLearn against [13] and [20], and demonstrate that TrajDistLearn has **0.11%** and **23.3%** more accuracy for the Character Trajectory dataset, and **1.99%** and **1.97%** more accuracy for the GISCUP dataset with Relative Distance $\leq 0.5\ unit$.

*Speed.* For two polygonal trajectories $P$ and $Q$ with $p$ and $q$ segments, respectively, the approach by Alt and Godau et al. [4], which is the traditional and naive method for computing Fréchet distance has a time complexity of $O(pq\log(pq))$ [4]. Therefore, it is dependent on the length of the trajectory in terms of the number of locations or segments that form the trajectory. Similar is true for other Fréchet distance approximation techniques like Eiter et al. [21] or Dutsch et al. [20]. Experiments validate that time for rasterization is trivial, and therefore, our results for this evaluation can further be split into two major categories:

- ***Independent of the length of the trajectory***. We evaluate our approach using trajectories with varying sizes (number of locations) for both datasets. As expected [4] shows an increase in the compute time, with an increase in number of locations. We also evaluate TrajDistLearn against other Fréchet distance approximation techniques like Dutsch et al. [20] and Eiter et al. [21]. We measure CPU Time for our approach on Intel(R) Xeon(R) E5-2683 and the GPU time on NVIDIA GEFORCE GTX 1080 Ti. As shown in Figure 4, our proposed approach has near-constant time performance irrespective of the size of the trajectory when compared to the other optimal Fréchet distance approximation techniques. TrajDistLearn rasterizes all trajectories, and thereby compute time is solely dependent on the image resolution. Image representation is downsampling in one form, but not explicit downsampling of trajectory data, unlike all the other approaches.
- ***Better overall speed for distance calculation***. We perform experiments with TrajDistLearn on both datasets and multiple system configurations. Table 6, compares the mean and standard deviation of compute time of TrajDistLearn against all the other Fréchet distance calculation and approximation techniques [4, 21, 26]. Our experiments prove that TrajDistLearn shows atleast a 3000x speed-up on GPU architectures and at least 40x speed-up on CPU architectures for Fréchet distance computation compared to the traditional pure distance computation method [4]. We also validate that TrajDistLearn can easily be parallelized and run on GPU architectures, and it shows a substantial

| | Relative Distance | | | | | |
|---|---|---|---|---|---|---|
| Distance Metric | $\leq 0.1$ | | $\leq 0.25$ | | $\leq 0.50$ | |
| Dataset | Character | GISCUP | Character | GISCUP | Character | GISCUP |
| Frechet | 65.84% | 61.60% | 94.74% | 87.12% | 98.76% | 100.00% |
| Discrete Frechet | 61.14% | 60.80% | 91.96% | 86.82% | 97.24% | 100.00% |
| Hausdorff | 52.30% | 61.32% | 89.36% | 84.50% | 97.44% | 100.00% |

**Table 3: Relative distance values for Character Trajectory and GISCUP dataset for Fréchet , Discrete Fréchet and Hausdorff distance computation using TrajDistLearn.**




**(a) Character Trajectory dataset**

**(b) GISCUP dataset**

**Figure 4: Impact of length of trajectory on the compute time for each of the Fréchet distance computation and approximation methods. The two Y-Axis represent the compute time for the two baseline methods and how they increase with the trajectory length. This plot shows how the compute time for TrajDistLearn remains nearly unchanged, even with a significant increase in the trajectory length, unlike the other methods.**

| Dataset | Relative Distance | | | $\Delta_{\%\leq 0.50}$ |
|---|---|---|---|---|
| | $\leq 0.1$ | $\leq 0.25$ | $\leq 0.50$ | |
| Character | 93.20% | 98.30% | 98.65% | **-0.11%** $\downarrow$ |
| GISCUP | 97.10% | 97.49% | 98.01% | **-1.99%** $\downarrow$ |

**Table 4: Relative distance values for Character Trajectory and GISCUP dataset for Fréchet distance computation using Bringman et al [13].The last column shows that [13] is less accurate as compared to TrajDistLearn.**

| Dataset | Relative Distance | | | $\Delta_{\%\leq 0.50}$ |
|---|---|---|---|---|
| | $\leq 0.1$ | $\leq 0.25$ | $\leq 0.50$ | |
| Character | 35.74% | 54.99% | 75.46% | **-23.3%** $\downarrow$ |
| GISCUP | 97.09% | 97.52% | 98.03% | **-1.97%** $\downarrow$ |

**Table 5: Relative distance values for Character Trajectory and GISCUP dataset for Fréchet distance computation using Dutsch et al [20].The last column shows that [20] is less accurate as compared to TrajDistLearn.**

speed-up compared to other Fréchet distance approximation techniques.

*Trajectory Similarity.* There has not been too much work using neural networks for solving the problem of trajectory similarity computation [22, 30, 43]. Experiments validate that apart from the harder problem of trajectory distance computation, TrajDistLearn can also solve this prevalent trajectory similarity computation application. Figure 5 proves the correlation of cosine similarity for trajectory features generated by our network and the actual distance calculated between those trajectory pairs using [4, 21, 26]. Hence, these experiments prove that cosine similarity among trajectory features can be effectively used as a metric to measure similarity among trajectories with TrajDistLearn.

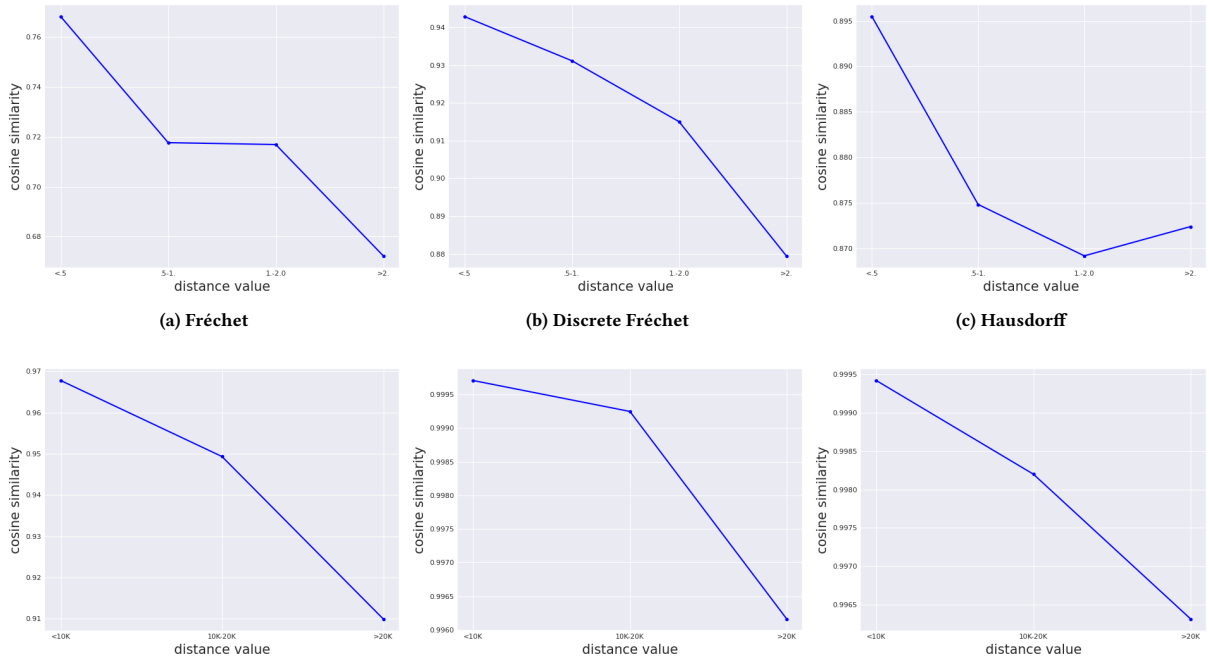
## 5 ABLATION STUDY

### 5.1 Rasterization

Now we discuss the impact of rasterization density on relative distance. As an illustration, we use Fréchet distance for both datasets. For all our experiments, we use a rasterization density of $R = 6$

| Dataset | Naive Fréchet Method [4] | | Discrete Fréchet [21] | | TrajDistLearn (CPU) | | TrajDistLearn (GPU) | |
|---|---|---|---|---|---|---|---|---|
| | Mean | Std.Dev | Mean | Std.Dev | Mean | Std.Dev | Mean | Std.Dev |
| Character | 6.60 | 0.78 | 0.015 | 0.0008 | 0.15 | 0.001 | **0.0020** | 3.28e-05 |
| GISCUP | 25.67 | 19.58 | 0.017 | 0.015 | 0.21 | 0.008 | **0.0027** | 9.46e-05 |

**Table 6: The compute time mean and standard deviation for Fréchet distance computation shows that TrajDistLearn outperforms traditional pure distance computation method [4] on both CPU and GPU. Also, TrajDistLearn is the only approach that can be trivially parallelized and run on GPU.**




**(a) Fréchet** **(b) Discrete Fréchet** **(c) Hausdorff**

**(d) Fréchet** **(e) Discrete Fréchet** **(f) Hausdorff**

**Figure 5: The figures represent the relation of cosine-similarity as a trajectory similarity measure and the distance among the trajectory pairs. (a-c) Character Trajectory dataset. (d-f) GISCUP dataset. The plots prove how using the features from TrajDistLearn and cosine-similarity among those features can effectively compute trajectory similarity.**

*pixels/bit*. Figure 6 illustrates the impact of substantial variation in rasterization density on relative distance for Fréchet distance computation for both datasets. We validate that the accuracy of our approach remains nearly unchanged, even with a major change in rasterization density while changing it from $R/2$ uptil $8R$.

## 5.2 Training Data Size

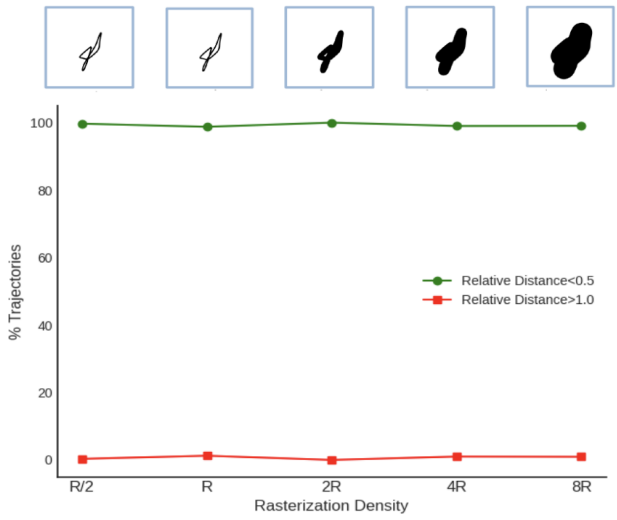
We evaluated the effect of training data size on relative distance. As an illustration, we evaluate the effect of training split sizes on Fréchet distance computation for both datasets and show in Table 7. We observe that even with a minimal training data size, our network can learn the distance function really well and lead to very high accuracy.

| | Relative Distance $\leq 0.5$ | |
|---|---|---|
| Training Split | Character | GISCUP |
| 10% | 95.46% | 100.00% |
| 70% | 97.46% | 100.00% |
| 90% | 98.76% | 100.00% |

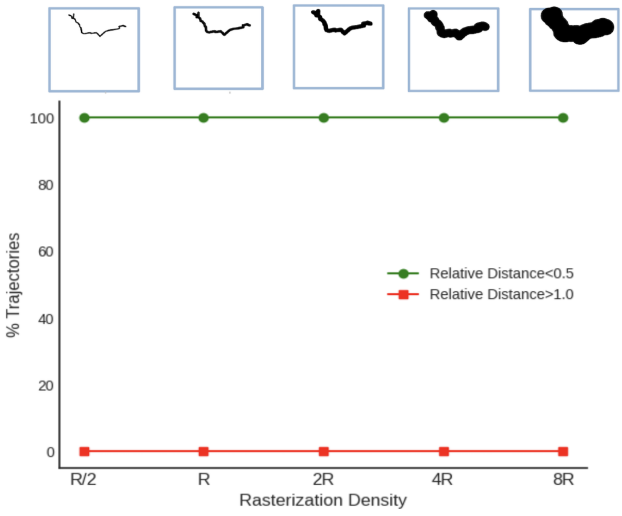
**Table 7: Effect of training data size on relative distance for Fréchet distance computation for both datasets. We observe little or no change in the accuracy based on the training data size.**

## 5.3 Depth of Network

Experiments validate that with increasing depth of the neural network for the siamese twins, we do not observe a change in distance measurement accuracy. As an illustration, we measure the impact




(a) Character Trajectory dataset




(b) GISCUP dataset

**Figure 6: Impact of rasterization on relative distance for Fréchet distance computation. The plot validates that TrajDistLearn is not sensitive to the rasterization density used for trajectory representation, as the accuracy remains nearly unchanged.**

of network depth on relative distance for Fréchet distance computation for the GISCUP dataset. The results of the experiment are shown in Table 8. We observe that the accuracy for distance approximation remains nearly unchanged, or slightly drops with an increasing depth of the neural network. Our explanation for a deeper network not impacting accuracy is the low dimensionality of the underlying location data and its representation as rasterized images.

| Network | Relative Distance $\leq 0.5$ |
|---|---|
| TrajDistLearn | 100.00% |
| TrajDistLearn + Conv | 95.96% |
| TrajDistLearn + 2xConv | 95.72% |

**Table 8: Impact of the neural network's depth for each of the siamese twins on relative distance for Fréchet distance computation for GISCUP dataset.**

| Dataset | Distance Metric | | | | | |
|---|---|---|---|---|---|---|
| | Frechet | | Dis Frechet | | Hausdorff | |
| | Batch Size | | Batch Size | | Batch Size | |
| | 8 | 32 | 8 | 32 | 8 | 32 |
| Character | 98.76% | 97.42% | 97.24% | 95.86% | 97.44% | 96.32% |
| GISCUP | 100.00% | 100.00% | 99.94% | 100.00% | 99.96% | 100.00% |

**Table 9: Effect of changing batch size while training on accuracy with Relative Distance $\leq 0.5$. The accuracy for distance computation does not change significantly with a change in the batch size.**

## 5.4 Batch Size

We evaluated our approach by experimenting with different batch sizes for training and their overall impact on the performance, i.e. training time and relative distance values. We look at the percentage of trajectory pairs with Relative Distance $\leq 0.5$ for both datasets and all distance metrics for ease of understanding.

As represented in Table 9 we generally observed that increasing the batch size while training did not lead to a significant change in the accuracy and also led to an increase in the per epoch training time and, thereby, an increased overall training time.

## 6 CONCLUSION

In this paper, we address the computationally complex problem of trajectory distance computation. TrajDistLearn is the first deep learning-based trajectory distance computation framework for all distance metrics that introduces a unique approach for representing trajectories as rasterized images (raw rasterized images and map images) also shows the ability to solve the task of trajectory similarity computation. The essential characteristics of TrajDistLearn lie in the fact that it is a very intuitive, robust, easy to train and easy to scale neural network architecture, such that we can use the same network parameters for all the distance metrics and all datasets. This work sheds light on several research directions: 1) Enhanced feature representation to capture various attributes of a trajectory like direction, time, intensity(or speed), and other possible physical attributes which would enable better feature learning. 2) Efficient trajectory space partitioning for improved performance on the convolution operation. 3) Using indexing techniques like Locality-Sensitive Hashing [5] to pre-process the trajectories. 4) Using spatial embeddings from this approach for other downstream applications like [38]. 5) Extending the proposed approach to localized trajectory search based on a set of meta-information.

## 7 ACKNOWLEDGEMENT

This work was sponsored in part by the NSF under Grants IIS-18-16889, IIS-20-41415, and IIS-21-14451.

## REFERENCES


[1] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. 2013. Computing the Discrete FrÉChet Distance in Subquadratic Time. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 156–167. http://dl.acm.org/citation.cfm?id=2627817.2627829

[2] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. 2002. Near-Linear Time Approximation Algorithms for Curve Simplification. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA '02)*. Springer-Verlag, London, UK, UK, 29–41. http://dl.acm.org/citation.cfm?id=647912.740828

[3] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. 2003. Matching Planar Maps. *J. Algorithms* 49, 2 (2003), 262–283.

[4] Helmut Alt and Michael Godau. 1995. Computing The Fréchet Distance Between Two Polygonal Curves. *Int. J. Comput. Geom. Appl.* 05, 1-2 (1995), 75–91. https://doi.org/10.1142/S0218195995000064

[5] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117.

[6] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. 2015. Fréchet Distance for Curves, Revisited. *European Symposium Algorithms* abs/1504.07685 (2015). arXiv:1504.07685 http://arxiv.org/abs/1504.07685

[7] Julian Baldus and Karl Bringmann. 2017. A fast implementation of near neighbors queries for Fréchet distance (GIS Cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 1–4.

[8] S. Belongie, J. Malik, and J. Puzicha. 2002. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 4 (2002), 509–522.

[9] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.

[10] P. C. Besse, B. Guillouet, J. M. Loubes, and F. Royer. [n. d.]. Review and Perspective for Distance-Based Clustering of Vehicle Trajectories. *IEEE TITS '16* ([n. d.]). https://doi.org/10.1109/TITS.2016.2547641

[11] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On Map-matching Vehicle Tracking Data. In *Proceedings of the 31st International Conference on Very Large Data Bases (PVLDB '05)*. 853–864. http://dl.acm.org/citation.cfm?id=1083592.1083691

[12] Karl Bringmann. 2014. Why Walking the Dog Takes Time: Fréchet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS '14)*. 661–670.

[13] Karl Bringmann, Marvin Künnemann, and André Nusser. 2019. Walking the Dog Fast in Practice: Algorithm Engineering of the Fr\'echet Distance. *arXiv preprint arXiv:1901.01504* (2019).

[14] Karl Bringmann and Wolfgang Mulzer. 2015. Approximability of the Discrete Fréchet Distance. In *31st International Symposium on Computational Geometry (SoCG 2015) (SoCG '15)*. 739–753.

[15] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*. 737–744.

[16] Kevin Buchin, Maike Buchin, and Yusu Wang. 2009. Exact Algorithms for Partial Curve Matching via the Fréchet Distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 645–654. http://dl.acm.org/citation.cfm?id=1496770.1496841

[17] Erin Wolf Chambers, Eric Colin De Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. [n. d.]. Homotopic Fréchet Distance Between Curves or, Walking Your Dog in the Woods in Polynomial Time. *SoCG '08* ([n. d.]). https://doi.org/10.1016/j.comgeo.2009.02.008

[18] Anne Driemel, Sariel Har-Peled, and Carola Wenk. 2010. Approximating the Fréchet Distance for Realistic Curves in Near Linear Time. In *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry (SoCG '10)*. 365–374.

[19] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[20] Fabian Dütsch and Jan Vahrenhold. 2017. A filter-and-refinement-algorithm for range queries based on the Fréchet distance (GIS Cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 1–4.

[21] Thomas Eiter and Heikki Mannila. 1994. *Computing discrete Fréchet distance*. Technical Report. Technische Universitat Wie.

[22] Michael R Evans, Dev Oliver, Shashi Shekhar, and Francis Harvey. 2013. Fast and exact network trajectory similarity computation: a case-study on bicycle corridor planning. In *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*. ACM, 9.

[23] Atlas F. Cook Iv and Carola Wenk. 2008. *Geodesic Fréchet Distance With Polygonal Obstacles*. Technical Report.

[24] Maurice Fréchet. 1914. *Sur quelques points de calcul fonctionnel*.

[25] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*. 47–57.

[26] Felix Hausdorff. 1914. *Grundzuge der Mengenlehre*.

[27] Felix Hausdorff. 1927. *Mengenlehre*. Walter de Gruyter Berlin.

[28] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

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

[30] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 617–628.

[31] Javier Ribera, David Güera, Yuhao Chen, and Edward Delp. 2018. Weighted Hausdorff Distance: A Loss Function For Object Localization. *arXiv preprint arXiv:1806.07564* (2018).

[32] Hanan Samet. 1983. A quadtree medial axis transform. *Commun. ACM* 26, 9 (1983), 680–693.

[33] Hanan Samet. 1985. Reconstruction of quadtrees from quadtree medial axis transforms. *Computer vision, graphics, and image processing* 29, 3 (1985), 311–328.

[34] J. Sankaranarayanan and H. Samet. 2009. Distance Oracles for Spatial Networks. In *2009 IEEE 25th International Conference on Data Engineering (ICDE '09)*. 652–663. https://doi.org/10.1109/ICDE.2009.53

[35] Jagan Sankaranarayanan and Hanan Samet. 2010. Roads Belong in Databases. *IEEE Data Eng. Bull.* 33 (2010), 4–11.

[36] Jagan Sankaranarayanan and Hanan Samet. 2010. Roads belong in databases. (2010).

[37] Divya Shanmugam, Davis Blalock, and John Guttag. 2018. Jiffy: A Convolutional Approach to Learning Time Series Similarity. (2018).

[38] Hong Wei, Janit Anjaria, and Hanan Samet. 2019. Learning Embeddings of Spatial, Textual and Temporal Entities in Geotagged Tweets. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 484–487.

[39] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. 2013. Map Matching: Comparison of Approaches Using Sparse and Noisy Data. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. 444–447.

[40] Martin Werner and Dev Oliver. 2018. ACM SIGSPATIAL GIS Cup 2017: Range queries under Fréchet distance. *SIGSPATIAL Special* 10, 1 (2018), 24–27.

[41] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 3880–3887.

[42] Sook Yoon, Hyouck Min Yoo, Sang Hoon Yang, and Dong Sun Park. 2010. Computation of discrete Fréchet distance using CNN. In *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*. IEEE, 1–6.

[43] Hanyuan ZHANG, Xingyu ZHANG, Qize JIANG, Baihua ZHENG, Zhenbang SUN, and Weiwei SUN. 2020. Trajectory similarity learning with auxiliary supervision and optimal matching.(2020). In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Yokohama, Japan*. 11–17.