Dual Supervised Autoencoder Based Trajectory Classification Using Enhanced Spatio-Temporal Information

SICHONG LU AND YING XIA
Chongqing Spatial Big Data Intelligent Engineering Research Center, School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
Corresponding author: Ying Xia (xiaying@cqupt.edu.cn)
This work was supported in part by the Natural Science Foundation of China under Grant 41571401 and Grant 41971365; and in part by the Natural Science Foundation of Chongqing, China, under Grant cstc2019jcyj-msxm1096.

ABSTRACT With the rapid development of mobile internet and location awareness techniques, massive spatio-temporal data is collected every day. Trajectory classification is critically important to many real-world applications such as human mobility understanding, urban planning, and intelligent transportation systems. A growing number of studies took advantage of the deep learning method to learn the high-level features of trajectory data for accurate estimation. However, some of these studies didn’t interpret spatio-temporal information well, more importantly, they didn’t fully utilize the high-level features extracted by neural networks. To overcome these drawbacks, this paper utilizes the proposed stop state and turn state to enhance spatial information, and at the same time, extracts stronger time information via Recurrence Plot (RP). Moreover, a novel Dual Convolutional neural networks based Supervised Autoencoder (Dual-CSA) is proposed by making the network aware of Predefined Class Centroids (PCC). Experiments conducted on two real-world datasets demonstrate that Dual-CSA can learn the high-level features well. The highest accuracy of the Geolife and SHL datasets are 89.475% and 89.602%, respectively, proving the superiority of our method.


I. INTRODUCTION
Nowadays, the wide availability of mobile internet makes trajectory data easy to acquire. Various positioning techniques or devices such as mobile phones, on-board Global Position System (GPS), wearable devices have much better accuracy and performance, yield us reliable trajectory data. Understanding and discovering knowledge from trajectory data is a key issue in urban computing [1]. In real life, numerous applications are based on trajectory data mining techniques such as effectively identifying users’ transportation modes [2]–[9], analysis on vessel activities [10]–[13], traffic congestion prediction [14]–[16], stampede events warning [15], and abnormal trajectory detection [17], [18]. For research purposes, studies like learning animal migratory habits and identifying the intensity of hurricane [19] are highly relying on trajectory data mining techniques. These applications can help people manage the city, improve public safety, avoid natural disasters, and make the environment sustainable. Accordingly, we can infer that trajectory classification plays a crucial role in these applications. Various moving objects have diverse behavior patterns, but the original trajectory data cannot directly reflect these differences. Therefore, it is necessary to utilize data mining techniques to distinguish the trajectories of different categories. In this paper, we mainly focus on classifying urban transportation modes into five classes, including walk, bike, bus, driving (taxi and car), and train (or subway).

In the previous decade, trajectory classification has been studied extensively and achieved good results. Many of them exploited feature engineering to extract the motion characteristics of trajectory, including velocity, acceleration, heading change rate, and stop rate [20], [21], which can be further fed into classical machine learning methods to estimate the class of trajectory. Although there is rich spatio-temporal information inside motion characteristics, many previous studies only...
focused on one of them, and rarely considered both, mainly because the features they used were statistical (e.g., mean velocity and expectation of velocity). Lately, deep learning has frequently been used in trajectory classification tasks and achieved better performance. Deep learning is capable of automatically discovering the representations of raw input, in other words, high-level features are progressively extracted [22]–[24]. Dabiri and Heaslip [8] mainly built a network based on Convolutional Neural Networks (CNN) to extract high-level features from motion characteristics and used a Softmax layer as a classifier. Liu and Lee [5] proposed a network based on Bidirectional Long Short Term Memory (Bi-LSTM) that directly learn the high-level features from trajectory coordinates and timestamps, and the Softmax layer was also used. Endo et al. [9] developed a method that combined high-level features and handcrafted features, in which, the high-level features were learned from trajectory images they generated using an autoencoder. Unlike the previous two methods, they used classical machine learning methods such as Logistic Regression (LR) and Support Vector Machine (SVM) as classifiers. Nevertheless, due to the incompressibility in latent space, these studies did not make full use of high-level features, which should be treated as a latent distribution and make breakthroughs from it.

To address the above drawbacks, similarity to previous studies, motion characteristics are also adopted, which are named Movement Features (MFs) in this paper. Considering the behavior of moving objects in space, we propose Auxiliary Features (AFs) to augment spatial information. Meanwhile, time information inside MFs and AFs is augmented by calculating RPs. Afterward, we propose Dual-CSA, which has the ability to make use of high-level features as much as possible in latent space.

The main contributions of this paper are summarized as follows:

1) AFs such as stop state and turn state are proposed to enrich the spatial information. To effectively interpret the time information inside MFs and AFs, we develop a method to encode them as RPs.

2) We propose a cluster centroids aware supervised autoencoder that can make the high-level features (latent embedding) of the same class of samples as close as possible to PCC, which is mainly benefited from a loss function we improved from the one proposed in [25], and this improvement is based on the Pre-defined Evenly-Distributed Class Centroids (PEDCC) algorithm [26].

3) A neural network consisting of two autoencoders is designed, one of which learns the high-level features of RPs and the other learns the high-level features of AFs and MFs. We also develop a method to fuse these high-level features by concatenating them. Moreover, a two-stage training including pretraining and joint training is proposed, in particular, the joint training is applied a dynamic schedule.

The remainder of this paper is organized as follows. Related works are reviewed in Section II. Section III declares definitions, formally presents the problem of this study, and briefly introduces some of the techniques used in this paper. Following that, in Section IV, we introduce the framework of trajectory classification based on transportation modes and detail the Dual-CSA. In Section V, the results of conducted experiments along with experimental settings are reported, and ablation studies are presented. Section VI discusses the time efficiency and sampling rate sensitivity of the model. Finally, Section VII concludes the paper and presents future work.

II. RELATED WORKS

Trajectory classification is a ubiquitous and fundamental task in many trajectory data mining applications. In the last few years, trajectory classification has attracted extensive attention from researchers. One of the representative studies is classifying transportation modes using various data sources such as GPS trajectories [2]–[9], GPS-based check-in trajectories, mobile phone’s GSM data [27], and accelerometers data [28]. Besides, there are also some researches about vessel type identification using Automatic Identification System (AIS) trajectories [10]–[12], detecting anomaly behavior using video object trajectories [29], and air traffic flow characterization using radar tracks [30]. Above mentioned studies may require more than one sensor, nonetheless, in many circumstances, GPS is the only available data source, hence, in this section, we mainly review the studies that used GPS trajectories or other coordinates based trajectories as data source. Literature [31] provided a comprehensive and systematic review from the perspective of point-based classification and segment-based classification. Considering features play an important role in most of these works, we divide the existing trajectory classification methods into three categories: point-level features based, segment-level features based, and non-feature based.

At the beginning of the trajectory classification study, segment-level features were one of the frequently used features. For instance, Zheng et al. [20] as one of the trailblazers in this area of research, proposed to partition trajectory into segments based on the domain knowledge that people must stop when changing the transportation mode, each of which corresponding a unique transportation model. Certain hand-crafted features like length and mean velocity can be easily extracted for each segment and were then used in several classical machine learning methods for mode classification. Following this line of work, more studies have emerged. A recent study by Zhu et al. [3] found that the trajectory segmentation method based on the change point was not accurate. For example, it is impossible to switch the transportation mode of “bus → walk → bus → walk → bus” in a very short distance. In their work, this issue was well solved by merging adjacent short segments. In terms of mining new segment-level features, they introduced time slice type (rush hour or not) to deal with the problem that driving is
estimated as walking in rush hours because of the low speed. More recently, Nawaz et al. [32] further improved the aforementioned segmentation algorithm. In their study, instead of straightforward merging short segments with adjacent segments, short segments were considered in context specifically to decide whether to merge or not. In particular, they also took efforts to developed segment-level features, especially one of them named transport mode probabilities (one for each transportation mode), which yielded good performance. Throughout these prior segment-based studies, we can infer that good features devote a lot in the subsequent classification algorithm, and in turn, classification algorithm should also be chosen appropriately. Usually, more than one algorithm was tested in their experiments, such as k-Nearest Neighbor (kNN), SVM, and Decision Tree (DT).

Indeed, the aforementioned segment-level features are derived from statistical analysis of point-level features, which have also been extensively studied. Especially when it comes to real-time scenes, point-level features have better performance than segment-level features. For example, in 2009, Byon et al. [33] proposed an approach that can infer travelers’ transportation modes within the real-time GPS data stream. Specifically, as GPS data came in, variables sampled in time including speed, acceleration, number of satellites in view, and electromagnetic levels, which were further fed into the Multilayer Perceptron (MLP) to determine the transportation mode. It’s worth mentioning that this study also confirmed the potential of neural networks to successfully detect traffic modes from GPS data. And in recent years, contributed by the recent breakthrough in deep learning, a growing body of literature has attempted to learn much higher features from point-level features in a deep learning way. A follow-up research by Dabiri and Heaslip [8] referred to such an approach as integrating the hand-crafted features (point-level features) with high-level features. The proposed method did follow this principle, that is, movement features including speed, acceleration, jerk, and bearing rate were calculated between every two consecutive points, and then using a specially designed network based on CNN to learn the high-level features from point-level features in a deep learning way. A follow-up research by Dabiri et al. [34] introduced roadway features to combine with movement features, including the number of intersections, number of maneuvers, and road type. All features of a GPS leg (the route segment between two consecutive GPS points) were stacked into a vector for later input into the CNN network. In both studies above, point-level features they developed were very helpful and effective, based on which, Nawaz et al. [35] proposed region index (map latitude and longitude into an index) as a new feature to aid point-level features. The authors adopted convolutional LSTM (ConvLSTM) to implement their architecture due to its ability to address both the spatial and temporal dependencies at the same time.

In contrast to point-level features or segment-level features, some methods don’t rely on any predefined features. Cuenca et al. [36] studied trajectory classification based on Volunteer Geographic Information (VGI). VGI is low resolution and uncertainty, therefore, the authors introduced a classifier based on fuzzy rules to handle this kind of data. Their fuzzy rule classifier didn’t involve any features but required some trajectory data mining techniques before generating fuzzy rules, including trajectory clustering and trajectory patterns mining. Liu et al. [5] proposed a model based on Bi-LSTM. Instead of extracting the trajectory features in advance, their model directly took the trajectory (i.e., longitude and latitude) as the input of the network. Timestamp as another important information in GPS data was also passed into the network. Consequently, their model can capture spatial and temporal features. The above two studies share a similarity that they both kept the original form of trajectory data and passed it to the classifier. However, some studies managed to transform the trajectory into another forms. Endo et al. [9] proposed a method to generate trajectory images, in particular, they transformed the trajectory segments into grid images and learned high-level features from them using an autoencoder. They also extracted handcrafted features that were used to concatenated with high-level features for further passing into classical machine learning algorithms. Wang and Oates [37] also proposed a method to encode the trajectory data as images. By regarding trajectory as time series and considering trajectory points are multidimensional, they converted trajectory data into a one-dimensional time series using the Hilbert curve [38], which were then encoded into images using the proposed Gramian Angular Field (GAF) and Markov Transition Field (MTF), thus high-level features can be learned using their tiled CNN model.

However, compared to the method proposed in this paper, the above methods have at least one of the following constraints. Firstly, some methods needed external data (e.g., number of satellites) to aid classification, which is not always available. Therefore, this paper only uses GPS data but still can achieve high accuracy. Secondly, many methods used segment-level features which are essentially local global features, some spatial information details have lost. So in this paper, point-level features are adopted. Thirdly, it is hard for deep learning based methods to interpret both spatial and temporal information well, hence, many methods only considered spatial or temporal information (e.g., encoding trajectory data as images only considers spatial information). Fourthly, many studies used deep learning to learn high-level features, which are not fully utilized (e.g., simply feed them into the Softmax layer).

To tackle the above shortcomings, we propose an approach that using a dual CNN based supervised autoencoder to classify trajectory based on transportation modes from only GPS data, which utilizes point-level features and augments these features with AF and RP. As PEDCC is capable of aiding classification in latent space, we introduce the PCC layer, which is also an important component of the
Dual-CSA. In general, our method falls into the point-level feature based category, more specifically, it integrates the point-level feature with high-level features.

### III. PRELIMINARIES

#### A. DEFINITIONS AND PROBLEM STATEMENT

We first present the formal notion related to trajectory classification and then give the problem statement.

**Definition 1 (Trajectory):** A trajectory is a sequence of points, denoted as \( TR = \{ p_1, p_2, \ldots, p_n \} \). Each point \( p \) is a tuple of latitude, longitude, and timestamp, denoted as \( p = (x, y, t) \), which determines the geographic location at time \( t \).

**Definition 2 (Segment):** A segment is one of the partitioned results of a trajectory, represented as \( SE = \{ p_1, p_2, \ldots, p_m \} \), where \( m \) is the number of segment points, and each segment only corresponds to one class.

**Definition 3 (Segment Label):** Segment label \( l \in O \) is a label that indicates which class that segment belongs to, where \( O \) is a set of all classes (e.g., \( O = \{ \text{Bus, Walk, Train, Bike}, \} \)).

**Definition 4 (Change Point):** Suppose there is a moment in a \( TR \) where the movement behavior changes (e.g., the transportation mode changes from bus to walk), the point at this moment is named change point.

**Definition 5 (Feature Segment):** A feature segment is calculated from an \( SE \), each element of which is a value of a point feature, denoted as \( FS = \{ f_1, f_2, \ldots, f_{n-1} \} \), where \( f \) can be any point feature such as velocity, acceleration, and heading change rate.

Based on these definitions, the formal trajectory classification problem is defined as follow:

**Problem 1:** Given a training dataset \( D \) of \( SEs \), by first calculating on each \( SE \) to get an \( FS \) or \( FSs \) (presented as a tensor for deep learning architectures) along with a label \( l \), and further having an entire feature-label set \( \mathcal{F} = \{(FS_1, l_1), (FS_2, l_2), \ldots, (FS_n, l_n)\} \), we aim to learn high-level features from \( \mathcal{F} \) and build a classifier to estimate the class of an \( SE \).

#### B. MOVEMENT FEATURES OF SEGMENTS

The fact that different moving objects usually have diverse moving behaviors makes MFs play an important role in trajectory data mining, as well as in the trajectory classification task. In Table 1, MFs used in this paper are summarized. For consecutive points in an \( SE \), Fig. 1 depicts how we calculate the MFs. The time difference between \( p_i \) and \( p_{i+1} \) is denoted as \( \Delta t_i \). We can calculate velocity \( v_i \) and acceleration \( a_i \) using the following equations:

\[
\begin{align*}
    v_i &= \text{Vincenty}(p_i, p_{i+1})/\Delta t_i, \\
a_i &= (v_{i+1} - v_i)/\Delta t_i,
\end{align*}
\]

where \( \text{Vincenty}(\cdot, \cdot) \) [39] is used to calculate the geographical distance. Heading direction \( h_i \) is the angle between the north-south line of earth and the line connecting two consecutive points, which can be calculated as:

\[
\begin{align*}
    X &= \cos(x_{i+1}) \cdot \sin(y_{i+1} - y_i), \\
    Y &= \cos(x_i) \cdot \sin(x_{i+1}) \cdot \cos(y_{i+1} - y_i), \\
    h_i &= \text{atan2}(X, Y).
\end{align*}
\]

Heading change rate \( hcr_i \) is calculated as:

\[
hcr_i = (h_i - h_{i-1})/\Delta t_i.
\]

#### C. RECURRENCE PLOT

RP was first introduced in 1987 by Eckmann et al. [40]. It is often used to visualize the recurrences of dynamical systems, enabling us to find the periodic properties of trajectories in phase space in a visual way. To better understand the recurrence plot, we first state the concept of phase space reconstruction. Considering the time series as a nonlinear dynamic system, it is hard to discover the multi-dimension property of the system due to the original time series only contain scalar. For the purpose of reconstructing the original dynamic system from time series, delay coordinate embedding [41] is utilized. Given a time series:

\[
\{x_1, x_2, \ldots, x_N\},
\]

for this single observable \( x \), reconstruction of the unobserved and possibly multi-dimensional phase space dynamic is governed by two parameters, embedding dimension \( m \) and time delay \( \tau \) [42]. The resultant phase space vector set is:

\[
Y = [y_1, y_2, \ldots, y_n]^T,
\]

where \( n = N - (m - 1) \tau \) and

\[
y_k = (x_k, x_{k+\tau}, x_{k+2\tau}, \ldots, x_{k+(m-1)\tau}),
\]

for \( k = 1, 2, \ldots, m \).

---

**TABLE 1. Summary of MFs.**

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity</td>
<td>10 m/s</td>
</tr>
<tr>
<td>acceleration</td>
<td>2 m/s²</td>
</tr>
<tr>
<td>heading change rate</td>
<td>10°/s</td>
</tr>
</tbody>
</table>

**FIGURE 1. The illustration of MFs calculation.**
Takens [43] proposed the embedding theorem in 1981, this theorem gives us a guarantee that phase space equivalent to the original dynamical system topologically from a one-dimensional time series always can be reconstructed if \( m \geq 2\tau + 1 \), note that it is a sufficient but not necessary condition for original dynamical system reconstruction.

Based on the phase space reconstruction mentioned above, RP is visualized through the matrix:

\[
R_{i,j} = \begin{cases} 
1 & \text{if } \|y_i - y_j\| \leq \varepsilon \\
0 & \text{otherwise} 
\end{cases},
\]

where \( y_i \) and \( y_j \) are the i-th and j-th vector in \( Y \), \( \varepsilon \) is a threshold distance, \( \| \cdot \| \) is a norm. RP yields important insights into the time evolution of these trajectories, because typical patterns in an RP are related to specific behaviour of the system [44]. The resultant matrix is binarized by the threshold parameter \( \varepsilon \). To avoid this, the thresholding step is skipped in this paper.

D. AUTOENCODER

Autoencoder was first proposed in [45] for the pretraining stage of a neural network, which is a feed-forward multi-layer neural network. Fig. 2 illustrates its basic structure, where the input \( x \) is mapped into latent embedding \( z \) via encoder \( e \), which in turn is mapped into reconstruction \( x' \) by decoder \( d \). The autoencoder is trained by a loss function such as Mean Square Error (MSE) to minimize the reconstruction error:

\[
L_r = \frac{1}{n} \| x - x' \|^2,
\]

where \( n \) is the number of input samples.

The learned latent embedding \( z \) is a dimensionally reduced representation of the original data, which can be considered as high-level features or the latent distribution of the original data.

IV. PROPOSED FRAMEWORK

A. FRAMEWORK OVERVIEW

Fig. 3 shows the detail of the proposed framework. We first partition the \( TR \) into \( SEs \) based on the change point of transportation mode [20], hence each \( SE \) corresponds to only one transport mode. Next, MFs for each \( SE \) are calculated, including velocity, acceleration, and heading change rate. To enrich the spatial information of MFs, AFs such as stop state and turn state are also extracted. As a result, we have a new data form that is presented as \( FSs \). In order to better extract the time information, we calculate RP for each \( FS \). Inspired by the PEDCC [26] algorithm and Deep Embedded Clustering (DEC) [25] model, we introduce a CNN based supervised autoencoder that can learn the high-level features well and leverage them as much as possible. Besides, we also make the network accept both \( FSs \) and RPs as inputs by extending the network to dual autoencoder, thus forming the Dual-CSA. Finally, after going through pretraining and joint training, the Dual-CSA can be used to trajectory classification.

B. TRAJECTORY SEGMENTATION

Since multiple classes can exist in a trajectory, we need to segment each \( TR \) into \( SEs \) based on the change point. As mentioned in the preliminaries, each \( SE \) is assigned with a label \( l \). Walking is selected as the segmentation criteria of change point based on the assumption that people must stop and walk when changing transportation modes [20]. Besides, \( SEs \) with too few points are discarded. In this paper, the minimum number of points in an \( SE \) is defined as 10.

C. ENHANCED SPATIO-TEMPORAL INFORMATION

Using the calculations of MFs introduced in the preliminaries, we calculate MFs for each \( SE \). However, according to
kinematics, velocity, acceleration, and heading change rate are all relative, they only reflect how fast the moving object changes its physical states. In other words, after calculating an MF from two consecutive coordinate points, the spatial information contained in these points is partially lost. Simply adding coordinate as another feature to enrich spatial information is deficient, because the semantic of coordinates is too weak, and has been proved the poor performance in our experiments during the research.

To tackle the above issue, the key is to enrich the spatial information for MFs with spatial semantics. For moving objects, stop and turn are the two most common states that can reflect spatial behavior well. Therefore, two AFs including stop state and turn state are proposed, which are formally defined as follows:

Definition 6 (Stop State): A stop state is a state indicating whether a moving object has stopped. Given two consecutive points \( p_i \in SE \) and \( p_{i+1} \in SE \), we use the following equation to calculate the stop state \( s_{i+1} \):

\[
s_{i+1} = \begin{cases} 1 & \text{if } \text{Vincenty}(p_i, p_{i+1}) \leq \lambda \\ 0 & \text{otherwise} \end{cases},
\]

where \( \lambda \) is a threshold distance. That is, if the distance between \( p_{i+1} \) and \( p_i \) is smaller than \( \lambda \), we assign 1 to \( p_{i+1} \), otherwise we assign 0.

Definition 7 (Turn State): Similar to the stop state, the turn state is a state that indicates whether a moving object is turning or not. Turn state \( u_{i+1} \) is computed by:

\[
u_{i+1} = \begin{cases} 0 & \text{if } |h_{i+1} - h_i| \leq \omega \\ 1 & \text{otherwise} \end{cases},
\]

where \( \omega \) is a threshold angle. This equation indicates that if the heading direction angle between \( p_i \in SE \) and \( p_{i+1} \in SE \) is smaller than \( \mu \), we assign 0 to \( p_{i+1} \), otherwise we assign 1.

In the end, we can get multiple FSs of MFs and AFs for each SE.

Dabiri and Heaslip [8] have proved that making FSs (point-level features) as the input of the neural network is a feasible approach. In this paper, we further augment the time information inside point-level features via RP. As detailed in preliminaries, RP is a modern method of nonlinear data analysis that can reflect periodicity in time series. By calculating RP for each FS, much more useful time information can be extracted. Fig. 4 visualizes the steps to calculate an RP. The curve in the left part of Fig. 4 is an acceleration FS extracted from an SE of the walk category. The right part of Fig. 4 is an RP calculated from FS. The texture in RP presents regularity, which is beneficial for CNN to learn texture features (detailed in the next section).

In addition, it is worth mentioning that the neural network usually requires the input of fixed size, and FSs of different sizes need to be aligned to the same size. If the input size accepted by the network is \( S \), for FS whose size is greater than \( S \), further segmentation is needed, for FS whose size is less than \( S \), we apply linear interpolation to them. After that, these FSs can be used to generate RPs. Since \( S \) determines the size of RP, and the size of RP should not be too large, or it will lead to a long time training. In this paper, we set \( S \) to 200, which ensures that the segment of this length has enough information, while the size of RP is not too large.

AF is used to enhance spatial information, however, it cannot play an auxiliary role if it works alone. In other words, AFs and MFs need to be combined in a proper way, so that the behavior of moving objects in space can be more perspicuous. Inspired by the channel construction method in [8], we combine the AFs and MFs in a similar way and make them adapt to the input of the network. Fig. 5 details the combination method. First, the FSs of AFs and MFs extracted from an SE are stacked into a volume with the shape of \( R^{1 \times 3 \times 5} \) (see right part of Fig. 5). Afterward, RPs of these FSs are calculated, and they are also stacked into a volume but with the different shape of \( R^{L \times L \times 5} \) (calculated based on (8) and (9)).

D. DUAL CNN BASED SUPERVISED AUTOENCODER USING PREDEFINED CLASS CENTROIDS

From the last section, we can conclude that AF and RP not only enhance spatio-temporal information but also have a better data structure to fit the neural network compared to the original trajectory data. Now the key is to design a neural network that can learn the texture features in RPs and the high-level features in FSs simultaneously. Since CNN is well known for having good performance on handling image data and sequence data, in this study, CNN is utilized to construct our neural network. As illustrated in Fig. 6, the proposed model consists of two autoencoders, named dual autoencoder. The input arrows in Fig. 6 indicate that the
The dual autoencoder accepts RPs and FSs as inputs, and thus these two autoencoders are named RP autoencoder and FS autoencoder, respectively. The RP encoder is composed of several convolutional layers and max pooling layers. Since RP has a small size, we make the convolution kernel has the shape of $R^{3 \times 3}$ and the filter for max pooling layer has the shape of $R^{2 \times 2}$. All these layers have a stride of 1. The end of the encoder is the flatten operation. As the decoder is the reverse process of the encoder, its hyperparameters are set to the same as the encoder. Similarly, for FS autoencoder, the only difference from RP autoencoder is that the height of filter or convolution kernel is set to 1 since AFs and MFs are 1-dimensional data. In the studies of natural language processing, such architecture is called 1D-CNN [46]. Finally, to better utilize the high-level features learned by dual autoencoder, we concatenate two latent embedding vectors to fuse them (see the purple dot box in Fig. 6).

The vanilla autoencoder is often used to learn a low dimension representation (high-level features) of original data. Directly using the low dimension embedding data as the high-level features may lead to low classification accuracy due to the training process of the vanilla autoencoder is unsupervised or auto-supervised. To tackle this, we create a classification layer called PCC layer, which is connected to the bottleneck (see the purple dot box in Fig. 6). The PCC layer can be regarded as a variant of the clustering layer proposed in [25], which maps each embedded point $z_i$ of input data $x_i$ into a soft label by Student’s t-distribution [47]:

$$q_{ij} = \frac{1}{\sum_j \left(1 + \|z_i - \mu_j\|^2\right)^{-1}} \left(1 + \|z_i - \mu_j\|^2\right)^{-1},$$  \hspace{1cm} (14)

where $q_{ij}$ is the $j$th entry of $q_i$, representing the probability of $z_i$ belonging to class $j$. $\mu_j$ is class centroids calculated using the PEDCC algorithm [26]. Apparently, $\mu_j$ and $z_i$ have the same shape of $R^{D \times 1}$, where $D$ is the size of the latent embedding dimension (the purple dot box in Fig. 6). For an $n$ classification problem, PEDCC can generate $n$ points that are evenly distributed on the $D$-dimensional hypersphere surface as clustering centers. PEDCC is an iterative process which mainly employs a physical model of lowest like charge energy...
on the hyperspherical surface. In this algorithm, the resultant force vector $f_i$ for $i$-th points can be calculated by:

$$f_i = \sum_{j=0}^{n} \frac{1}{|I_j|^2} I_j, \quad j \neq i,$$

(15)

where $I_j$ is the distance vector between two points. The position of each point $\mu_i$ will be updated in the iteration with the current velocity vector $v_i$. For $D$-dimensions embedding vectors, the algorithm generates $n$ points of $D$-dimensions predefined class centroids ($\mu_1, \ldots, \mu_n$). The algorithm ends until it reaches the maximum number of iterations.

The soft assignments $q_i$ need to match the ground truth distribution. To this end, the classification loss $L_c$ is defined as KL divergence between the soft assignments $q_i$ and the ground truth distribution $p_i$ (i.e., trajectory labels after one-hot encoding). It is formulated as follows:

$$L_c = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

(16)

The gradient of the classification loss function $L_c$ with respect to $z_i$ is given by:

$$\frac{\partial L}{\partial z_i} = \frac{\xi + 1}{\xi} \sum_j \left(1 + \frac{\|z_i - \mu_j\|^2}{\xi}\right)^{-1} \times (p_{ij} - q_{ij}) (z_i - \mu_j),$$

(17)

where $\xi$ are the degrees of freedom of the Student’s t-distribution, according to [25], we let $\xi = 1$ for all experiments.

Combining the reconstruction loss and classification loss, the objective of Dual-CSA can be defined as:

$$L = \alpha L_{r1} + \beta L_c + \gamma L_{r2},$$

(18)

where $L_{r1}$ and $L_{r2}$ are MSE loss functions (see (11)) of RP autoencoder and FS autoencoder respectively, and $\alpha$, $\beta$, $\gamma$ are used to weigh these three loss functions, which are referred to as “loss weights” in the rest of this paper.

Furthermore, if we denote the parameters (weights and bias) of FS encoder and RP encoder as $\eta$ and $\sigma$ respectively, the gradient of the loss function $L$ with respect to $\eta$ and $\sigma$ can be expressed as:

$$\nabla L(\eta) = \alpha \nabla L_{r1}(\eta) + \beta \nabla L_c(\eta),$$

(19)

$$\nabla L(\sigma) = \alpha \nabla L_{r2}(\sigma) + \beta \nabla L_c(\sigma),$$

(20)

since $\eta$ and $\sigma$ mainly affect the classification results, the above equations mean we can balance the gradient contributed by different parts of the network during the backward.

**E. MODEL TRAINING**

The training of Dual-CSA consists of two stages including pretraining and joint training. In the first stage, the pretraining is only performed on dual autoencoder. We expect the pretraining can learn an initial underlying distribution of the input data, thus enhancing the discrimination ability in the second stage. Since the pretraining is unsupervised training, we should join the PCC layer into the training at a proper time to avoid the underlying distribution becoming not helpful for classification, because pretraining too much will lead the network to be more inclined to reconstruct the input data rather than discrimination ability (see the analysis in part D of the next section). Therefore, we introduce joint training as the second stage that aims to extract the most useful latent data representations, and at the same time, enhance discrimination ability. Besides, we apply a schedule of dynamic loss weights to control the degree of distortion of the underlying distribution, that is, we change the value of $\alpha$, $\beta$, $\gamma$ in (18) over training epochs (detailed in part B of next section). Note that during the training, we choose Adam optimizer as our optimization technique, and use Leaky Rectified Linear Unit (Leaky ReLU) as the activation function, which overcomes dying neuron issue of ReLU. To keep the model from overfitting, we employ the early stopping method.

**V. EXPERIMENTS**

**A. EXPERIMENTAL SETUP**

1) EXPERIMENTAL SETTING

All experiments are conducted on two dedicated servers and their detail configuration is summarized in Table 2, in which, the pytorch is used to implement Dual-CSA and its variants, the keras is used to implement the network we use for comparison, and scikit-learn is used to implement data preprocessing and classical machine learning methods. Implementations for our framework can be found online at: https://github.com/lusccc/Trajectory-Classification-using-Dual-CSA

<table>
<thead>
<tr>
<th>Name</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon <a href="mailto:W-2133@3.6GHz">W-2133@3.6GHz</a> x 1, <a href="mailto:W-2123@3.6GHz">W-2123@3.6GHz</a> x 1</td>
</tr>
<tr>
<td>Memory</td>
<td>96GB</td>
</tr>
<tr>
<td>GPU</td>
<td>RTX 2080 Ti 11GB x 3, GTX 1070 Ti 8GB x 1</td>
</tr>
<tr>
<td>Operating system</td>
<td>Ubuntu 18.04</td>
</tr>
<tr>
<td>Language</td>
<td>Python v3.7.7</td>
</tr>
<tr>
<td>Development frameworks</td>
<td>Pytorch v1.6.0 [48], Keras v2.3.1 [49], scikit-learn v0.23.2</td>
</tr>
</tbody>
</table>
TABLE 3. The maximum value of velocity $v_{\text{max}}$, acceleration $a_{\text{max}}$, and heading change rate $hcr_{\text{max}}$ of different transportation modes.

<table>
<thead>
<tr>
<th>Transportation mode</th>
<th>$v_{\text{max}}$ (m/s)</th>
<th>$a_{\text{max}}$ (m/s$^2$)</th>
<th>$hcr_{\text{max}}$ (°/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walk</td>
<td>7</td>
<td>3</td>
<td>360</td>
</tr>
<tr>
<td>Bike</td>
<td>12</td>
<td>3</td>
<td>360</td>
</tr>
<tr>
<td>Bus</td>
<td>34</td>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>Driving</td>
<td>50</td>
<td>10</td>
<td>120</td>
</tr>
<tr>
<td>Train</td>
<td>34</td>
<td>3</td>
<td>30</td>
</tr>
</tbody>
</table>

Each value of $v_{\text{max}}$ and $a_{\text{max}}$ is borrowed from [8]. Note that values of $hcr_{\text{max}}$ are calculated based on empirical values. For example, the minimum turning radius of a car is generally 6.5 m. In urban roads, the maximum speed under this radius is about 50 km/h, and then the $hcr_{\text{max}}$ for car can be calculated as $t = (2 \times \pi \times 6.5)/(50/3.6), hcr_{\text{max}} = 360/t \approx 122.48^\circ/s$. And for walk and bike modes, the $hcr_{\text{max}}$ is set to 360° because users can turn around in a very short time.

April 2007 to August 2012). Among them, 73 users have labeled their trajectories with transportation modes, such as driving, taking a bus, riding a bike, and walking.

• SHL Dataset: The SHL dataset was collected in 2017 over a period of seven months in the UK. Three volunteers participated in this project, and each of them equipped three Huawei mate 9 mobile phones to record data such as movement, pressure, and GPS trajectory. Similar to the Geolife dataset, all these data were labeled with eight types of transportation mode such as still, walk, and run.

In this paper, we only consider ground transportation modes. Car and taxi are treated as the same class, denoted as driving. Similarly, train and subway are also referred to as the same class, denoted as train. Therefore, the set of all classes is \{Walk, Bike, Bus, Driving, Train\}. Dataset is randomly split into 8:2 as train dataset and test dataset respectively. Moreover, several preprocessing steps are applied, including:

1) Trajectory whose number of GPS points less than 10 is discarded.
2) GPS point whose timestamp is smaller than the previous one is discarded.
3) GPS point whose longitude or latitude is out of range is discarded.
4) According to Table 3, the values of MFs that exceed the maximum value of the corresponding transportation mode are discarded.

Fig. 7 summarizes the number of SEs for each transportation mode after preprocessing, which also shows us that the Geolife dataset has more samples than the SHL dataset but the data distribution is imbalanced, while the SHL dataset is completely opposite. At last, standardization for each feature of both MF and RP is required. The standard score of sample $x$ is calculated as

$$z = (x - u)/s,$$  \quad (21)

where $u$ is the mean of the training samples, and $s$ is the standard deviation of the training samples.

3) EVALUATION METRICS

In order to evaluate the performance of classification, we use accuracy, precision, recall, and F1 score as our evaluation metrics. They are defined as:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$ \quad (22)
$$\text{precision} = \frac{TP}{TP + FP},$$ \quad (23)
$$\text{recall} = \frac{TP}{TP + FN},$$ \quad (24)
$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$ \quad (25)

where $TP$ is true positives, $TN$ is true negatives; $FP$ is false positives; $FN$ is false negatives.

The confusion matrix is also employed to make the classification results more intuitive. It is an informative table used to analyze the errors and confusions between different classes. In this table, each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa) [53].

B. PARAMETER SETTING AND SCHEDULING

The primary parameters to be tuned in our proposed framework including embedding dimension $m$ and time delay $\tau$ in phase space reconstruction, embedding dimension $D$ in latent space, and the dynamic loss weights $\alpha, \beta, \gamma$. We first state how to determine $m$ and $\tau$. As has already been noted in preliminaries, the key to RP calculation is phase space reconstruction. The embedding dimension $m$ and time delay $\tau$ should be chosen properly to embody the nature of the original dynamic system. In this paper, we use the mutual information function [54] and false nearest-neighbours algorithm [55] to calculate the embedding dimension $m$ and time delay $\tau$, respectively. Specifically, we average the $m$ and $\tau$ calculated from all FSs of MFs and then round them up. In our experiment, $m$ and $\tau$ on both datasets yielded us very close values, i.e., $\tau = 7.30$ and $m = 2.91$ for Geolife dataset, ...

http://www.shl-dataset.org/download/
\( \tau = 7.79 \) and \( m = 2.33 \) for SHL datasets. After rounding up, we get \( m = 3 \) and \( \tau = 8 \) for both datasets, which is a very reasonable result considering both datasets share a similar nature.

The embedding vector generated by Dual-CSA in latent space should only keep the most salient high-level features, if its dimension \( D \) is too small, the useful information may be lost, and if \( D \) is too large, it will increase the training time. We determined \( D \) by performing a line search on the set \( \{8k | k \in [1..50]\} \) with \( m = 3 \), \( \tau = 8 \) and \( \alpha = \beta = \gamma = 1 \). To obtain reliable results, we repeated the experiment for various \( D \) 10 times and took the average accuracy. Note we only experimented on the SHL dataset due to the expensive time consuming if we experiment on both datasets. The results in Fig. 8 shows that \( D = 304 \) (the red column) works best.

The schedule of dynamic loss weights we mentioned earlier can automatically balance the weight of the dual autoencoder and PCC layer according to the validation score during the joint training. We experimented on multiple different schedules with other parameters fixed to the value acquired above, and found one described below outperforms others, which consists of two steps:

1) Setting \( \alpha = \gamma = \beta = 1 \): In the first step, we assign RP autoencoder, FS autoencoder, and PCC layer with the same weight. This means each part of the network can equally learn the most useful features from the data.

2) Gradually reduce \( \alpha \) and \( \gamma \) to 0 while keeping \( \beta \) unchanged: If the validation score not improved from the previous setting after 4 epochs, we reduce 0.1 for \( \alpha \) and \( \gamma \) while keeping \( \beta \) still. We repeat this process until \( \alpha = \gamma = 0 \). No improvement from the last step means the optimization has stuck in local minima, at this time, we should make the trade-off of the contribution among the different parts of the network. By reducing the effect of dual autoencoder, more gradient is contributed by the classification part (see (19) and (20)) make the network learn more helpful features for discrimination, also push the optimization out of the local minima to move towards better local minima.

Table 4 also presents the test accuracy on other configurations of schedules. \#1 sets \( \alpha = \gamma = \beta = 1 \) throughout the entire training is a naive approach and has a good performance, but still lower than \#4. We tried \#3 and \#4 with \( \beta = 32 \) to expect the network to put more emphasis on classification. However, although it gave us a fast convergence, it didn’t increase the accuracy.

After tuning the above parameters, we summarize all parameters or schedule as follows: \( m = 3, \tau = 8, D = 304, \) and dynamic loss weights from \( \alpha = \gamma = \beta = 1 \) to \( \alpha = 0, \beta = 1, \gamma = 0 \).

### C. EXPERIMENTAL RESULTS

**1) RESULTS OF USING OPTIMAL PARAMETERS**

Using the parameters identified from the last section, we present the best results as confusion matrices in Table 5 and Table 6 for both Geolife and SHL dataset, and the highest test accuracy reaches 0.89475 and 0.89602 respectively, showing the good performance of our model. For the Geolife dataset, the major classification errors occur between bike and walk modes, which are mainly caused by an imbalance sample number that the number of walk mode is almost twice that of bike mode (see Fig. 7), along with the similar moving behavior of walking and bike. Hence results in the low recall of bike but walk is still relatively high. Similar results also appear in the SHL dataset, the sample number of bus mode is much greater than train mode, and they also share a similar moving behavior, causes low precision and recall for both bus and train modes.

**2) COMPARATIVE RESULTS**

To demonstrate the strength of our model, a variety of competing methods are used to compare, which are grouped into two categories: classical machine learning methods, and deep learning based methods. All the experiments for each method were repeated 10 times and take the average results to present.

#### a: CLASSICAL MACHINE LEARNING METHODS (CML)

Before deep learning became popular, classical machine learning methods were widely adopted in trajectory classification studies, including RF, kNN, SVM, DT, and MLP. According to previous studies, the most common features they used are segment-level features. Specifically, segment-level features such as length, mean velocity, expectation of velocity, covariance of velocity, top three velocities,
TABLE 5. The best results of the confusion matrix for Dual-CSA framework on Geolife dataset. Accuracy = 0.89475.

<table>
<thead>
<tr>
<th>Dual-CSA</th>
<th>Predicted</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walk</td>
<td>2386</td>
<td>0.88534</td>
</tr>
<tr>
<td>Bike</td>
<td>205</td>
<td>0.89151</td>
</tr>
<tr>
<td>Bus</td>
<td>37</td>
<td>0.89331</td>
</tr>
<tr>
<td>Driving</td>
<td>53</td>
<td>0.93462</td>
</tr>
<tr>
<td>Train</td>
<td>14</td>
<td>0.88770</td>
</tr>
</tbody>
</table>

TABLE 6. The best results of the confusion matrix for Dual-CSA framework on SHL dataset. Accuracy = 0.89602.

<table>
<thead>
<tr>
<th>Dual-CSA</th>
<th>Predicted</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walk</td>
<td>172</td>
<td>0.96629</td>
</tr>
<tr>
<td>Bike</td>
<td>5</td>
<td>0.96528</td>
</tr>
<tr>
<td>Bus</td>
<td>0</td>
<td>0.83704</td>
</tr>
<tr>
<td>Driving</td>
<td>0</td>
<td>0.97647</td>
</tr>
<tr>
<td>Train</td>
<td>1</td>
<td>0.70536</td>
</tr>
</tbody>
</table>

and top three accelerations [20] were frequently adopted due to their effectiveness. In literature [21], improved and more robust features were put forward, including heading change rate, stop rate, and speed change rate. Using the built-in functions provided by the scikit-learn library of the above classical machine learning methods, we conduct experiments based on 13 segment-level features. The results are listed in Table 7, showing that the performance of our model exceeds them by a large margin, and the F1 score of each class indicates that classical machine learning methods encounter the issue of low F1 score in certain categories. Besides, the results on the SHL dataset are much lower than Geolife dataset, which means classical machine learning methods are sensitive to the data scale, while our model has similar performance on both datasets.

b: DEEP LEARNING BASED METHODS (DL)
Considering trajectory data is in fact time series, many neural networks have proven their abilities to learn temporal features effectively, such as Recurrent Neural Networks (RNN) and LSTM networks, which are deployed for comparison in this paper. Specifically, we implement them using the keras framework and mainly connecting a Softmax layer to the end of each network respectively. Recently, the LSTM Fully Convolutional Networks (LSTM-FCN) proposed by Karim et al. [56] has been proved to have a state-of-the-art performance on time series classification. Using its published code,3 we also experimented to compare with it. It is worth mention that the features we used for the above three networks are MFs and AFs since they only can handle sequence data. From the results presented in Table 7, we find that RNN and LSTM don’t work very well especially on train modes, the LSTM-FCN provided a relatively balanced F1 score and good accuracy, but still far behind our model. The results also mean that the existing time series classification networks cannot be directly used to handle trajectory data, and prove that it is necessary to consider both temporal and spatial information.

3) COMPARE WITH PREVIOUS STUDIES
In the past few years, numerous researches about trajectory classification have emerged, especially the methods based on deep learning. To make a fair comparison, we choose the studies using the same dataset and transportation modes as ours. Table 8 briefly summarizes each method along with the corresponding accuracy. The results show that the proposed method outperforms others by a large margin. Literature [8] has the closest accuracy to ours, but still lower than our method by 4.7%.

D. ABLATION STUDY
The proposed framework is composed of multiple parts and is affected by many factors. We need to evaluate the relative contribution of each part and find out how these factors affect performance. In this section, we test each part or factor using the optimal parameters identified before. All the experiments are repeated 10 times and presented as average values.

1) EFFECT OF PCC LAYER
We first evaluate our model by simply replacing the PCC layer with a Softmax classifier (a fully connected layer connected with a Softmax layer). This variant of our model is named Dual CNN based Autoencoder using the Softmax layer (Dual-CA-Softmax). As can be seen from the bottom category of Table 7, our Dual-CSA can improve the accuracy by 1%~ 2% on both datasets compared with

3https://github.com/titu1994/LSTM-FCN
2) EFFECT OF DUAL AUTOENCORDER

To show the effectiveness of each autoencoder in Dual-CSA, we evaluate our model by removing one of them, which gives us two variants of our model, named CNN based Supervised Autoencoder using RP (CSA-RP) and CNN based Supervised Autoencoder using Feature Segment (CSA-FS). The results are also listed in the bottom category of Table 7, showing that either CSA-RP or CSA-FS are inferior to Dual-CSA. Dual autoencoder significantly improves the accuracy by up to 3.2%, and improves the F1 score on most categories by up to 9.8%, proving that our model works best when time information and spatial information is combined.

3) EFFECT OF MF AND AF

To demonstrate the effectiveness of MF and AF, we evaluate our model under different combinations of MFs and AFs. As can be seen in Table 9, from top to bottom, there is a significant improvement in accuracy and F1 score as the number of features used increases. After adding stop state and turn state, both accuracy and F1 score are improved by 1% ± 2%.

4) EFFECT OF PRETRAINING AND JOINT TRAINING

At last, we evaluate our model using different training strategies. Table 10 presents different training strategies and their results, in which, the strategy of no pretraining and no joint training can be seen as dropping the decoders in Dual-CSA. The results show that our model works best when adopting both pretraining and joint training, and outperforms the other strategies by up to 1.1%. Besides, we also use t-distributed Stochastic Neighbor Embedding (t-SNE) [47] to visualize latent embedding during the training process. Fig. 9 is the visualization results of the Geolife dataset while different colors mark the different classes, which show that pretraining works well from the first forwarding (i.e., only embed to latent space) to the pretraining epoch as the samples belong to the same class become aggregated, until epoch 6, the latent space presented as an obvious tendency of distortion. At this time the autoencoder probably starts to learn the features that more helpful for reconstruction rather than classification. Therefore, we should manually set the pretraining epochs to 5, otherwise too much pretraining will lead to performance degradation.

**TABLE 7.** Performance comparison with competitive methods and variants of our network (detailed in ablation study), where Dual-CA-Softmax, CSA-RP, and CSA-FS are variants of Dual-CSA.

<table>
<thead>
<tr>
<th>Category</th>
<th>Features</th>
<th>Method</th>
<th>Dataset</th>
<th>F1 Score</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CML</td>
<td>13 segment-level features</td>
<td>RF</td>
<td>Geolife SHL</td>
<td>0.85058</td>
<td>0.82404</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kNN</td>
<td>Geolife SHL</td>
<td>0.75985</td>
<td>0.59621</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SVM</td>
<td>Geolife SHL</td>
<td>0.85785</td>
<td>0.76285</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DT</td>
<td>Geolife SHL</td>
<td>0.67945</td>
<td>0.64478</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MLP</td>
<td>Geolife SHL</td>
<td>0.75985</td>
<td>0.68600</td>
</tr>
<tr>
<td>DL</td>
<td>MFs+AFs</td>
<td>RNN</td>
<td>Geolife SHL</td>
<td>0.82084</td>
<td>0.62241</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTM</td>
<td>Geolife SHL</td>
<td>0.87986</td>
<td>0.76976</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTM-FCN</td>
<td>Geolife SHL</td>
<td>0.86051</td>
<td>0.72803</td>
</tr>
<tr>
<td>Ours</td>
<td>MFs+AFs+RPs</td>
<td>Dual-CA-Softmax</td>
<td>Geolife SHL</td>
<td>0.91052</td>
<td>0.82064</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSA-RP</td>
<td>Geolife SHL</td>
<td>0.89660</td>
<td>0.95804</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSA-FS</td>
<td>Geolife SHL</td>
<td>0.90115</td>
<td>0.81644</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dual-CSA</td>
<td>Geolife SHL</td>
<td>0.91674</td>
<td>0.83308</td>
</tr>
</tbody>
</table>

**TABLE 8.** Accuracy comparison with previous deep learning based studies.

<table>
<thead>
<tr>
<th>Literature</th>
<th>Method Overview</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dabiri et al. [8]</td>
<td>CNN architecture with a four-channel GPS segment (speed, acceleration, jerk, and bearing rate).</td>
<td>0.848</td>
</tr>
<tr>
<td>Endo et al. [9]</td>
<td>Use autoencoder to extract high-level features from generated trajectory images.</td>
<td>0.679</td>
</tr>
<tr>
<td>Zhang et al. [57]</td>
<td>Map trajectory into a grid of multi-scale time and space granularities. Use the attention mechanism and residual learning to build a neural network. See detail in Section IV</td>
<td>0.733</td>
</tr>
<tr>
<td>This paper</td>
<td>This paper</td>
<td>0.89475</td>
</tr>
</tbody>
</table>

Dual-CA-Softmax. The F1 score also improved in most categories, especially in the driving and train categories, the improvement even up to 4.1%. Based on these results, the PCC layer can be proved effective.

VOLUME 8, 2020

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature-No.</th>
<th>Dataset</th>
<th>Accuracy</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity</td>
<td>①</td>
<td>Geolife</td>
<td>0.78350</td>
<td>0.77345</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.76911</td>
<td>0.72508</td>
</tr>
<tr>
<td>acceleration</td>
<td>②</td>
<td>Geolife</td>
<td>0.69092</td>
<td>0.66313</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.79969</td>
<td>0.76841</td>
</tr>
<tr>
<td>heading change rate</td>
<td>③</td>
<td>Geolife</td>
<td>0.75313</td>
<td>0.74295</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.68960</td>
<td>0.68564</td>
</tr>
<tr>
<td>stop state</td>
<td>④</td>
<td>Geolife</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>turn state</td>
<td>⑤</td>
<td>Geolife</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>①+②</td>
<td>-</td>
<td>Geolife</td>
<td>0.82313</td>
<td>0.81969</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.85780</td>
<td>0.84603</td>
</tr>
<tr>
<td>①+③</td>
<td>-</td>
<td>Geolife</td>
<td>0.86779</td>
<td>0.86539</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.85211</td>
<td>0.80177</td>
</tr>
<tr>
<td>②+③</td>
<td>-</td>
<td>Geolife</td>
<td>0.84434</td>
<td>0.83845</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.85321</td>
<td>0.84496</td>
</tr>
<tr>
<td>①+②+③</td>
<td>-</td>
<td>Geolife</td>
<td>0.88290</td>
<td>0.88165</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.88226</td>
<td>0.87425</td>
</tr>
<tr>
<td>①+②+③+④</td>
<td>-</td>
<td>Geolife</td>
<td>0.89232</td>
<td>0.89120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.88838</td>
<td>0.89198</td>
</tr>
<tr>
<td>①+②+③+④+⑤</td>
<td>-</td>
<td>Geolife</td>
<td>0.89339</td>
<td>0.89243</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.89236</td>
<td>0.89337</td>
</tr>
</tbody>
</table>

TABLE 10. Performance comparison of different training strategies.

<table>
<thead>
<tr>
<th>Pretrain</th>
<th>Joint Train</th>
<th>Dataset</th>
<th>Accuracy</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Geolife</td>
<td>0.88835</td>
<td>0.88718</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.88807</td>
<td>0.88933</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Geolife</td>
<td>0.88280</td>
<td>0.88166</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.89083</td>
<td>0.89184</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Geolife</td>
<td>0.89339</td>
<td>0.89243</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHL</td>
<td>0.89236</td>
<td>0.89337</td>
</tr>
</tbody>
</table>

Deep learning is an important factor that makes Dual-CSA have good performance, however, deep learning based methods usually require longer training time, especially Dual-CSA has two-stage training. To present the efficiency of Dual-CSA, we list the training time (100 epochs) and the numbers of network parameters in Table 11. Besides, the method proposed by Dabiri and Heaslip [8] is used as a reference as it is a representative deep learning based study in this field. Note we only show the joint training time and accuracy for our model, since the method proposed by Dabiri et al has no pretraining stage. The results show that Dual-CSA has longer training and more parameters, but higher accuracy. This is mainly because RP is 2D data that requires more memory and computation. Nevertheless, the training time of CSA-FS is much shorter since the data that FS autoencoder used is 1D. This also indicates that if we only consider using CSA-FS, the training time can be shorter, the parameters can be fewer, but still have a high accuracy of 0.86172, which is higher than the method proposed in [8]. Despite the training time of Dual-SAE is longer, using the trained model to predict is very efficient. It took 1.25s on average to estimate 1000 SEs on our hardware, and each SE has 200 trajectory points.

Another issue worth discussing is whether the Dual-CSA is sensitive to the sampling rate of the trajectory. We test the trained model by random discarding points in a trajectory within the test dataset, and the discard percentage ranges from 10% to 80% with a step size of 10%. We illustrate the results of both Geolife dataset and SHL dataset in Fig. 10. The results indicate that the proposed Dual-CSA has good robustness to different sample rates, even the discard percentage is up to 80%, the accuracy is still above 70% for Geolife dataset.

VI. DISCUSSIONS

Compared with other methods, the proposed method has a significant improvement, which is mainly on the account of that we design the model in two perspectives: (1) Augment spatio-temporal information inside the trajectory; (2) Specifically design a neural network to accommodate spatio-temporal data. After taking these two aspects into consideration together, augmentation techniques we developed including AF and RP. Since they are 1D or 2D data structure, 1D-CNN and 2D-CNN are utilized. Moreover, as this study is a classification task, from this point of view, we enable the autoencoder aware of the predefined class centroids via the PCC layer. Finally, Dual-CSA has a good ability to incorporate spatio-temporal information and utilize high-level features.

In our experiments, the accuracy of the Geolife dataset dropped to 0.88083 if we set the pretraining epochs to 20. Similarly, we can identify the pretraining epochs of the SHL dataset and set it to about 100 epochs. Fig. 9(d) to Fig. 9(f) show that as the joint training epoch increases, the inner-class distance becomes closer and inter-class distance becomes larger, which demonstrates the effectiveness of joint training.

Deep learning is an important factor that makes Dual-CSA have good performance, however, deep learning based methods usually require longer training time, especially Dual-CSA has two-stage training. To present the efficiency of Dual-CSA, we list the training time (100 epochs) and the numbers of network parameters in Table 11. Besides, the method proposed by Dabiri and Heaslip [8] is used as a reference as it is a representative deep learning based study in this field. Note we only show the joint training time and accuracy for our model, since the method proposed by Dabiri et al has no pretraining stage. The results show that Dual-CSA has longer training and more parameters, but higher accuracy. This is mainly because RP is 2D data that requires more memory and computation. Nevertheless, the training time of CSA-FS is much shorter since the data that FS autoencoder used is 1D. This also indicates that if we only consider using CSA-FS, the training time can be shorter, the parameters can be fewer, but still have a high accuracy of 0.86172, which is higher than the method proposed in [8]. Despite the training time of Dual-SAE is longer, using the trained model to predict is very efficient. It took 1.25s on average to estimate 1000 SEs on our hardware, and each SE has 200 trajectory points.

Another issue worth discussing is whether the Dual-CSA is sensitive to the sampling rate of the trajectory. We test the trained model by random discarding points in a trajectory within the test dataset, and the discard percentage ranges from 10% to 80% with a step size of 10%. We illustrate the results of both Geolife dataset and SHL dataset in Fig. 10. The results indicate that the proposed Dual-CSA has good robustness to different sample rates, even the discard percentage is up to 80%, the accuracy is still above 70% for Geolife dataset.
datasets demonstrate the superiority of our model over several truth distribution. Extensive experiments on two real-world centroids by making soft assignments close to the ground classification to infer transportation modes. To overcome the issue of insufficient samples and imbalance in the dataset, we consider using data augmentation techniques.

### VII. CONCLUSION

Trajectory classification as a crucial role in trajectory data mining has attracted a lot of attention from researchers. More and more deep learning based trajectory classification methods have emerged, but most of them still cannot fully utilize the spatio-temporal information and high-level features. In this paper, a novel model named Dual-CSA is proposed, which is a neural network architecture for trajectory classification to infer transportation modes. To overcome the challenge of interpreting spatio-temporal information well, we first employ RP to enrich time information and develop AF to enhance spatial information. Then, a PCC layer is proposed to make the autoencoder aware of the predefined class centroids by making soft assignments close to the ground truth distribution. Extensive experiments on two real-world datasets demonstrate the superiority of our model over several compared methods. Furthermore, the analysis of effects on different components in the proposed model proves that PCC layer, dual autoencoder, MF, AF, pretraining, and joint training significantly improve the performance. Finally, the time efficiency and sampling rate sensitivity of the model are discussed.

As for feature work, LSTM will be used to improve our model due to its outstanding ability to capture time information from the data. To make the model more robust to noise, we will consider using denoising autoencoder [58] in the future.

### REFERENCES


