Semi-Supervised Deep Learning Approach for Transportation Mode Identification using GPS Trajectory Data

Sina Dabiri, Chang-Tien Lu, Kevin Heaslip, Chandan K. Reddy, *Senior Member, IEEE*

Abstract—Inferring the travel mode distribution of users is a fundamental step in several transportation fields such as travel demand analysis, transport planning, and traffic management. In this paper, we aim to identify users' transportation modes purely based on their GPS trajectories. A majority of studies have proposed mode inference models based on hand-crafted features, which might be vulnerable to traffic and environmental conditions. Furthermore, the classification task in almost all models have been performed in a supervised fashion while a large amount of unlabeled GPS trajectories has remained unused. Accordingly, for the first time, a deep convolutional semi-supervised architecture is proposed to not only automatically extract abstract features from GPS tracks but also exploit useful information in the unlabeled data. The network is comprised of a convolutional-deconvolutional autoencoder and a convolutional neural network for unsupervised and supervised learning, respectively. After converting raw GPS trajectories into efficient image-like tensors, the unsupervised and supervised components of the architecture are simultaneously trained. An optimum schedule for varying the balancing parameters between reconstruction and classification errors are also implemented. The proposed semi-supervised model, image-like tensor, and hyperparameter schedule are evaluated by comparing to several baselines and alternatives for various amounts of labeled data. Our experimental results demonstrate the superiority of the proposed model by over the state-of-the-art semi-supervised and supervised methods with respect to metrics such as accuracy and F-measure.

Index Terms—Deep learning; semi-supervised learning; convolutional neural network; convolutional autoencoder; GPS trajectory data; transport mode detection.

1 Introduction

The mode of transportation for traveling between two points of a transportation network is an important aspect of users' mobility behavior. Identifying the distribution of users' transportation modes is a key step towards many transportation fields including transport planning, transit demand analysis, auto ownership, and transportation emissions analysis. Traditionally, the information for modeling the mode choice behavior was obtained through travel surveys. High cost, low-response rate, time-consuming manual data collection, and misreporting are the main demerits of the survey-based approaches [1]. With the rapid-growth of ubiquitous GPS-enabled devices (e.g., smartphones), a constant stream of users' trajectory data can be recorded. A user's GPS trajectory is constructed by connecting GPS points of their GPS-enabled device. A GPS point contains the information of the device geographic location at a particular moment. Mining trajectory data, which contain rich spatiotemporal information regarding human activities, provokes several transport-domain applications such as incident detection, mobility pattern extraction, and transport mode inference [2]. Notwithstanding that several studies have integrated GPS trajectories with other information sources such as GIS and mobile phone's sensors for building their mode detection scheme [3], [4], we aim to predict a user's transportation mode purely based on their GPS trajectories.

A majority of models for learning transportation modes from GPS tracks consists of two steps: (1) extracting features from GPS logs, (2) feeding features to a supervised learning method for the classification task. Unlike many other data sources, the GPS-based trajectory does not contain explicit features for inferring transportation modes, which calls for feature-engineering. Much of the current literature has generated hand-crafted features using the descriptive statistics of motion characteristics such as maximum velocity and acceleration [5], [6], [7]. Hand-crafted features can range from simple features such as the proportions of accelerations and decelerations larger than 1 m/s^2 [8] to kurtoses and skewnesses of speed and acceleration [9]. After creating a pool of manual attributes, a wide range of traditional supervised mining algorithms has been used for performing the classification task including rule-based methods, fuzzy logic, decision tree, Bayesian belief network, multi-layer perceptron, and support vector machine [1].

However, the feature engineering not only requires expert knowledge but also involves biased engineering justification and vulnerability to traffic and environmental conditions. For example, one may use the maximum speed of a GPS trajectory as a discriminating feature. The immediate criticism is that the maximum velocity of a car might be equal to bicycle and walk modes in the congested condition. The other expert may choose the top three velocities and accelerations of the user's GPS trajectory as a potential solution for lack of information about traffic conditions. Nonetheless, another specialist might critique this solution by asking why not using the top four velocities or why not the minimum instead of maximum? Automated feature learning methods such as deep learning architectures is a remedy to the above-mentioned shortcomings. Deep learning algorithms are capable of extracting abstract representations in an automated manner and without human interference.

Our approach for extracting automated features from a GPS track is a deep learning framework based on Convolutional Neural Network (CNN). Accordingly, one of our main challenges is to convert the raw GPS trajectories into an adaptable layout for CNN schemes. Inspiring by the work [10], first the basic motion characteristics of every GPS point in a trajectory including relative distance, speed, acceleration, and jerk are computed. This results in generating a sequence of numerical features for every type of motion characteristic. Next, the computed motion sequences are concatenated to create an image-like GPS-trajectory tensor, where every sequence is equivalent to a channel in an RGB image. Such an image-like tensor not only yields a standard arrangement for the CNN scheme but also describes the kinematic motion of a transport mode. Furthermore, in contrast to the hand-designed approaches, our proposed image-like tensor involves all GPS points of a user's trajectory rather than a small subset in the hand-designed approaches such as GPS points with maximum velocity or acceleration. Next, the transportation mode of a GPS trajectory is inferred by training a CNN-based deep learning architecture on the image-like GPS trajectories.

Moreover, almost all research on travel mode inference has built their models using only labeled trajectories. Nonetheless, a significant portion of trajectories in a dataset has not been annotated since the acquisition of labeled data is a more expensive and labor-intensive task in comparison with collecting unlabeled data. Using the unlabeled data in addition to the labeled ones gives room to capture more properties of the data distribution, which allows for a further improvement on decision boundaries and a better generalization on unseen records [11]. Thus, our main objective in this paper is to improve the CNN classifier by leveraging the power of deep unsupervised learning algorithms such as Convolutional AutoEncoder (Conv-AE). Thus, a semi-supervised framework comprised of Conv-AE and CNN classifier is proposed. Both components are simultaneously trained by minimizing a cost function that is a linear combination of unsupervised and supervised losses. Tuning the hyperparameters that connect these losses is the most challenging part of our training procedure.

The key contributions of this work are summarized as follows:

- *Converting a raw GPS trajectory into an efficient image-like tensor* so as to automatically extract abstract features from GPS data using deep learning architectures. Our proposed layout involves all GPS points of a trajectory with salient motion characteristics of a moving object.
- *Proposing, for the first time, a deep semi-supervised convolutional architecture* to leverage both labeled and unlabeled GPS trajectories for predicting transportation modes. Furthermore, the structure of our semi-supervised framework is unique on its own.
- *Development of various joint and disjoint strategies* for training our proposed semi-supervised neural model.
- *Proposing an effective schedule for tuning balancing parameters* of the loss function in the joint training strategy, which outperforms other schedules used in the literature of deep semi-supervised frameworks.
- *Demonstrating the superiority of our image-like GPS arrangement, semi-supervised architecture, and schedule for varying hyperparameters* by comparing with several alternatives and baselines.

The rest of this paper is organized as follows. Existing works on both mode detection methods and deep semi-supervised schemes are listed in Section 2. After providing some preliminaries in Section 3, details of our framework including the creation of the input, development of the proposed semi-supervised model along with the training strategies are explained in Section 4. Our experimental results are reported in Section 5. Finally, the paper is concluded in Section 6.

2 Related work

To date, several studies have deployed various data sources (e.g., GPS, mobile phone accelerometers, and Geographic Information System) or a combination of them for inferring users' transportation modes [4], [12]. However, along with the scope and purpose of this study, we review the studies that have utilized only GPS data for designing a mode inference model. A comprehensive and systematic review of existing techniques for travel mode recognition based on GPS data is available in [1]. The paper provides an excellent comparison of various approaches in three categories including GPS data preprocessing, trip/segmentation identification, and travel mode detection. After reviewing GPS-based inference models, we briefly discuss on semi-supervised deep learning architectures for other applications.

2.1 GPS-Based Mode Inference Models

As mentioned, feature extraction and classification are two major tasks in the GPS-based mode inference frameworks. Since the classification is often performed by traditional supervised algorithms (e.g., support vector machine and decision trees), the feature-extraction designation is the primarily discerning factor among various mode detection frameworks.

In two seminal studies by Zheng et al. [5], [6], a solid framework upon hand-crafted features was proposed. After dividing a trip into segments with distinct transportation modes, a set of manual yet robust features were identified and fed into machine learning algorithms (e.g., decision trees) for the classification task. Features were divided into basic and robust groups. The basic group primarily contain descriptive statistics of velocity and acceleration of all GPS points while the robust category is composed of the heading change rate, stop rate, and the velocity change rate. They demonstrated that the robust features are less vulnerable to traffic conditions; however, using a combination of basic and robust features results in higher accuracy. Xiao et al. [7] generated new features by computing more descriptive statistics such as mode and percentile. The number of features were further augmented by introducing local features through profile decomposition algorithms. Menp et al. [9] found that spectral features of speed and acceleration are significantly effective based on statistical tests while auto- and cross-correlations, kurtoses, and skewnesses of speed and acceleration were not useful. Notwithstanding that research on semi-supervised mode inference is really scarce, Rezaie et al. [13] performed a semi-supervised label propagation method, yet based on a limited number of hand-crafted features including speed, duration and length of a trip, as well as the proximity of a trip start and end points to the transit network.

A small body of literature has sought to integrate hand-crafted and automated features that are extracted using deep neural networks [14], [15], [10]. After converting a raw GPS trajectory into an image-like matrix, a type of deep learning algorithm is employed to obtain high-level representations for the classification task. Nonetheless, to the best of our knowledge, none of the deep learning approaches for mode detection has been built on the top of both labeled and unlabeled trajectories.

2.2 Semi-Supervised Deep Learning Approaches

Semi-supervised frameworks based on deep learning algorithms (e.g. recurrent and convolutional neural networks and autoencoders) have been exploited for a variety of tasks, mainly in computer vision and natural language processing fields [16], [17]. A large volume of existing literature on semi-supervised deep-learning architectures falls into two major groups: (1) Two-step process, in which the network is first trained in an unsupervised fashion as the pre-training step, and then the supervised component of the model is tuned using the labeled data. (2) Joint process, in which both the unsupervised and supervised components (i.e., the whole network) are concurrently trained.

One technique for the pre-training phase is to sequentially train layers [18]. Each layer is pre-trained with an unsupervised learning algorithm such as autoencoders and Restricted Boltzman Machine as a separated block while its input is the output of the previous layer. Indeed, the layer-wise unsupervised training strategy helps optimization by finding a good initial set of weights near a good local minimum, which sets the stage for a final training phase [19]. In the next stage, the deep architecture is fine-tuned by performing a local search from a sensible starting point. Another pre-training strategy is to train the network as a whole, rather than greedy layer-wise training, and then fine-tune the model using weights obtained from the first phase as an initialization. A sequence to sequence model, for example, can be first trained in an unsupervised manner. This is achieved by training the sequence autoencoding, which simply replaces the output sequence in the sequence to sequence framework with the input sequence [20], [21]. Using the obtained weights in the first step as a starting point for the supervised learning gives rise to a better stabilization and generalization of the model.

As mentioned, one of the main objectives in the pre-training step is to prepare a good initialization for the supervised training and avoid a poor generalization of the model. However, with advances in initialization and regularization schemes for deep learning architectures [22], [23], pre-training strategies are getting replaced with joint training strategies. The core idea in the joint strategies is to optimize a hybrid loss function, that is a combination of unsupervised and supervised components, with the goal of simultaneously preserving reconstruction and discrimination abilities. While a type of classifier (e.g., multi-layer perceptron or softmax function) forms the supervised part, variants of autoencoders have been widely utilized for performing the unsupervised task. Variational autoencoders [11], convolutional-deconvolutional autoencoders [24], autoencoders based on a Ladder network [17], and recursive autoencoders [25] are typical examples of autoencoders that have been used in deep semi-supervised networks. A substitute for autoencoders is to add an entropy regularization upon unlabeled data into the supervised loss function. At each training strategy, the unlabeled data are annotated using the updated weights in the previous iteration [26]. A balancing parameter can be used in the hybrid loss function to trade off the supervised and unsupervised parts of the objective function [25], [24]. In Section 5, the effect of balancing parameter(s) in the ultimate performance of the model is examined.

3 PRELIMINARIES

This section introduces the preliminaries required to comprehend the proposed framework in Section 4. First, the mode detection problem is described. Afterward, we show how to compute the motion characteristics of each GPS point and hand-crafted features introduced in [5], [6]. It is should be noted that the described hand-crafted features are only used in baselines as a part of our model evaluation. Table 1 provides the notations that are used for the problem formulation and the proposed semi-supervised framework.

TABLE 1: Notations used in this paper.

Name	Description
p	GPS point
T	GPS trajectory
SE	GPS segment (one Sample)
M	Number of GPS points in SE
Y	Set of labels (transportation modes)
RD	Relative distance
Δt	Time interval
A	Acceleration
J	Jerk
BR	Bearing rate
A	Acceleration
$\mathbf{X}_l$	Image-like tensor for labeled SE
$\mathbf{X}_{ul}$	Image-like tensor for unlabeled SE
$\mathbf{X}_{comb}$	Combined $\mathbf{X}_l$ and $\mathbf{X}_{ul}$
$\hat{\mathbf{X}}$	Image-like tensor for reconstructed SE
f	Encoder function in autoencoder
g	Decoder function in autoencoder
h	Latent representation in autoencoder
P_l	Predicted probability distribution of $\mathbf{X}_l$
P_{ul}	Predicted probability distribution of $\mathbf{X}_{ul}$
Y_l	One-hot true-label vector of $\mathbf{X}_l$
Y_{ul}	Pseudo predicted probability distribution of $\mathbf{X}_{ul}$

3.1 Problem Formulation

The raw GPS trajectory of a user is represented as a sequence of time-stamped GPS points $p \in T$, $T = [p_1, \ldots, p_N]$. Each GPS point p is a tuple of latitude, longitude, and time

$p = [lat, lon, t]$, which identifies the geographic location of point p at time t. P is divided into trips if the time interval between two consecutive GPS points exceeds a pre-defined threshold [5]. Also, the user might commute with more than one transport mode in a single trip. For instance, one may travel to work by first driving to a parking lot, then taking a bus, and finally walking toward their workplace. As a result, T is partitioned into multiple segments when the transportation mode changes. Each GPS segment SE is a sample in the dataset and denoted as $SE = [p_1, \cdots, p_M]$, where M is the number of GPS points that forms SE. Every SE is traveled by a unique transportation mode $y \in Y$, where Y is a set of transportation modes such as bike and car. Thus, the mode inference problem is a multi-class classification that seeks for predicting the correct transportation mode for a SE. However, the raw SE needs to be transferred to an appropriate format before feeding into a machine learning algorithm. Either a set of hand-crafted features or a proper layout, represented as **X**, needs to be designed for using in traditional machine learning or deep learning architectures, respectively. The first step for performing either approach is to compute the motion characteristics of GPS points in each SE, which is described in the next section.

Definition 1 (Mode inference problem) Given the training data $\{(\mathbf{X}_i, y_i)\}_{i=1}^{n}$ for n samples SE_i, the mode inference problem is defined as building the optimal classifier that minimizes the prediction error. The trained model is then deployed to estimate a user's transport mode while traveling in transportation networks based on extracted features **X** from their GPS trajectories.

3.2 Motion Characteristics of GPS Points

Several motion features can be computed for every GPS point based on their geographic coordinates and timestamps. Given two consecutive GPS points in a SE (e.g., p_1 and p_2), their relative distance is computed using the widely-used Vincenty's formula [27]. Having the relative distance and time interval, other basic kinematic motions for p_1 are calculated based on the following equations:

$$RD_{p_1} = Vincenty(p_1[lat, lon], p_2[lat, lon]) \tag{1}$$

$$\Delta t_{p_1} = p_2[t] - p_1[t] \tag{2}$$

$$S_{p_1} = \frac{RD_{p_1}}{\Delta t_{p_1}} \tag{3}$$

$$A_{p_1} = \frac{S_{p_2} - S_{p_1}}{\Delta t_{p_1}} \tag{4}$$

$$J_{p_1} = \frac{A_{p_2} - A_{p_1}}{\Delta t_{p_1}} \tag{5}$$

where RD_{p_1}, Δt_{p_1}, S_{p_1}, A_{p_1}, and J_{p_1} represent the relative distance, time interval, speed, acceleration/deceleration, and jerk of the point p_1, respectively. Jerk, the rate of change in the acceleration, is a significant factor in safety issues such as critical driver maneuvers and passengers' balance in public transportation vehicles [28]. Jerk has been used in mode detection models for the first time in [10].

The rate of change in the heading direction of different transportation modes varies. For example, cars and buses have to move only alongside existing streets while people with walk or bike modes alter their directions more frequently [6]. Bearing rate is a motion attribute for quantifying the heading change among modes. As depicted in Fig. 1, bearing measures the angle between the line connecting two successive points and a reference (e.g., the magnetic or true north). The bearing rate is the absolute difference between the bearings of two consecutive points, which are calculated as follows:

$$\begin{aligned} y &= \sin\left(p_2[lon] - p_1[lon]\right) \times \cos\left(p_2[lat]\right) \\ x &= \cos([p_1[lat]) \times \sin([p_2[lat]) - \sin(p_1[lat]) \\ &\quad \times \cos(p_2[lat]) \times \cos\left(p_2[lon] - p_1[lon]\right) \\ Bearing_{p_1} &= \arctan(y, x) \\ BR_{p_1} &= \mid Bearing_{p_2} - Bearing_{p_1} \mid \end{aligned} \tag{6}$$

where sin, cos, and arctan are trigonometric functions. lat and lon need to be in radians before passing to Eq. (6). Analogously, the above-mentioned formulas are utilized to calculate the motion features of other GPS points in a SE.

Next, we identify and remove erroneous GPS points that have been generated due to several error sources such as satellite or receiver clocks. Every GPS segment is filtered by the following data processing steps:

- A GPS point with the timestamp greater than its next GPS point is identified and discarded.
- For labeled trajectories, a GPS point whose speed and/or acceleration do not fall within a certain and realistic range of its transportation mode, provided in Table 2, is identified and discarded.
- For unlabeled trajectories, due to lack of knowledge on transport modes, any GPS point of a segment that its speed and/or acceleration fall more 1.5 times the interquartile range above the third quartile or below the first quartile is identified and discarded. The interquartile range is the difference between the third and first quartiles of all speeds in a GPS segment.
- After removing the unrealistic GPS points, a segment with (1) the number of GPS points, (2) the total distance, or (3) total duration less than specified thresholds are identified and discarded.

3.3 Hand-Crafted Features

After calculating motion features of every $p \in SE$ using Eqs. (1-6), a set of hand-crafted features for a GPS SE can be computed as follows based on the definitions in [5], [6]:

- Total Length: $L = \sum_{i=1}^{M} RD_i$
- Mean Speed: $\bar{S} = \frac{L}{(t_M - t_1)}$
- Expectation Speed: $E(S) = \frac{\sum_{i=1}^{M} S_i}{M}$
- Variance of Speeds: $Var(\{S_1, \cdots, S_M\})$

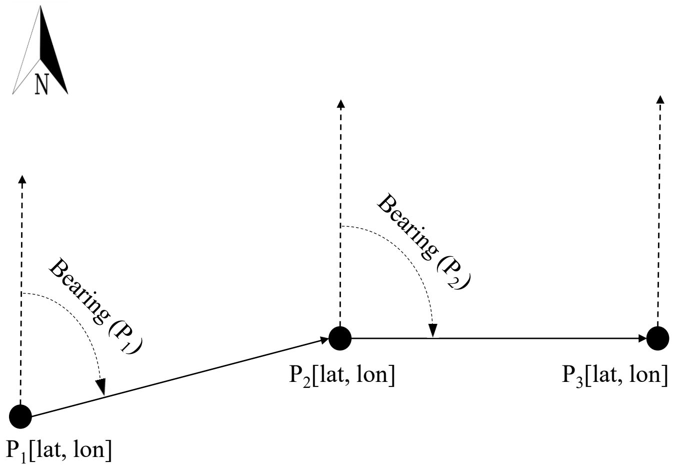

Fig. 1: Bearing between two consecutive GPS points

- Three Maximum Speeds from: $\{S_1, \cdots, S_M\}$
- Three Maximum Accelerations from: $\{A_1, \cdots, A_M\}$
- Heading change rate: $\mid p_c \mid /L$, where p_c is the number of GPS points in the SE that their Bearing exceeds a certain threshold.
- Stop Rate: $\mid p_s \mid /L$, where p_s is the number of GPS points in the SE that their speed is below a certain threshold.
- Speed Change Rate: $\mid p_v \mid /L$, where p_v is the number of GPS points in the SE that their speed rate exceeds a certain threshold.

The above-defined manual features are then fed into a supervised algorithm for performing the classification task. It should be noted that we do not use these features in our framework.

4 The Proposed Model

In this section, first, an efficient layout for each GPS segment is designed to be used as the input layer in our proposed framework. Next, the structure of our semi-supervised framework is elaborated. Finally, an effective strategy for training our network is proposed.

4.1 Creating Input Layer from Raw GPS Segments

Since the core component of our proposed framework is CNN, we need to convert GPS segments into a format that is not only compatible for CNN architectures but efficient enough to represent fundamental motion characteristics of a moving object. The motion features utilized in this study are relative distance (RD), speed (S), acceleration (A), and jerk (J). For every type of motion feature, a sequence can be created by putting the corresponding value for every GPS point of a SE in a chronological order, where the feature value is computed based on Eqs. (1-6). Such a sequence can be seen as a 1-d channel. Stacking these channels turns a raw GPS SE into a 4-channel image-like matrix. In Section 5, we demonstrate the superiority of such a configuration compared to other possible feature combinations.

Segments vary in the number of GPS points, which violates the CNN requirement. As a consequence, we either truncate long segments or pad short ones into a pre-defined number of GPS points, which in turn all segments are restricted to a fixed size. According to our observation, using the median size of all GPS segments as the fixed size results in having the best performance. In short, a raw GPS SE is mapped into a 4-channel image-like matrix with the shape of $(1 \times M \times 4)$, where M is the median of all GPS-segment sizes. Figure 2 illustrates the 4-channel structure for a GPS SE. Note that our proposed layout utilizes the information of all GPS points and allows the very algorithm to extract the efficient features for the mode inference. This is in contrast to the feature-engineering methods that leverage the information of a limited subset of the GPS segment such as points with the maximum speed and acceleration. Finally, values of each channel are individually scaled into the range [0,1] using the min-max normalization.

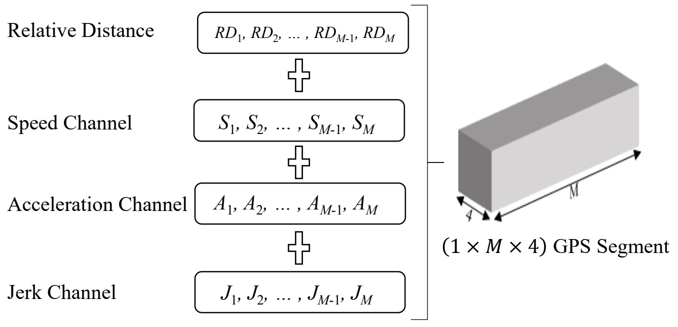
Relative Distance
$RD_1, RD_2, \ldots, RD_{M-1}, RD_M$
Speed Channel
$S_1, S_2, \ldots, S_{M-1}, S_M$
Acceleration Channel
$A_1, A_2, \ldots, A_{M-1}, A_M$
Jerk Channel
$J_1, J_2, \ldots, J_{M-1}, J_M$
M
4
$(1 \times M \times 4)$ GPS Segment

Fig. 2: A 4-channel image-like GPS segment with the shape of $(1 \times M \times 4)$

It is worth mentioning that our rationale for using CNN rather than recurrent neural network (RNN) has two-fold: (a) The speed, acceleration, and jerk of each GPS point, which are equivalent to each step in RNN, contain the time and distance information of its adjacent point(s) as well. Thus, nearby GPS points have already exchanged the motion information. (b) Since the traffic conditions and roadway geometry vary along a trip route, the chance of the change in mobility behavior of a single mode at different sections of a route increases. Accordingly, we are more interested in capturing the local correlation between GPS points in smaller sections while knowing that the motion information has already been passed between adjacent points.

4.2 Semi-Supervised Model for Transportation Mode Identification

As can be seen in Fig. 3, our semi-supervised architecture is composed of two main components: (1) a CNN classifier, which is fed by only labeled trajectories, denoted as $\mathbf{X}_l \in \mathbb{R}^{(1 \times M \times 4)}$, and (2) Convolutional-deconvolutional AutoEncoder (Conv-AE), which is fed by combined labeled and unlabeled trajectories, denoted as $\mathbf{X}_{comb} = (\mathbf{X}_l + \mathbf{X}_u) \in \mathbb{R}^{(1 \times M \times 4)}$. $\mathbf{X}_l$ and $\mathbf{X}_u$ are labeled and unlabeled GPS segments created based on the procedure described in the previous section.

4.2.1 *Convolutional-Deconvolutional AutoEncoder*

Autoendocer is an unsupervised learning technique that aims to learn an efficient latent representation by reproducing its input at the output layer. Autoencoder consists of

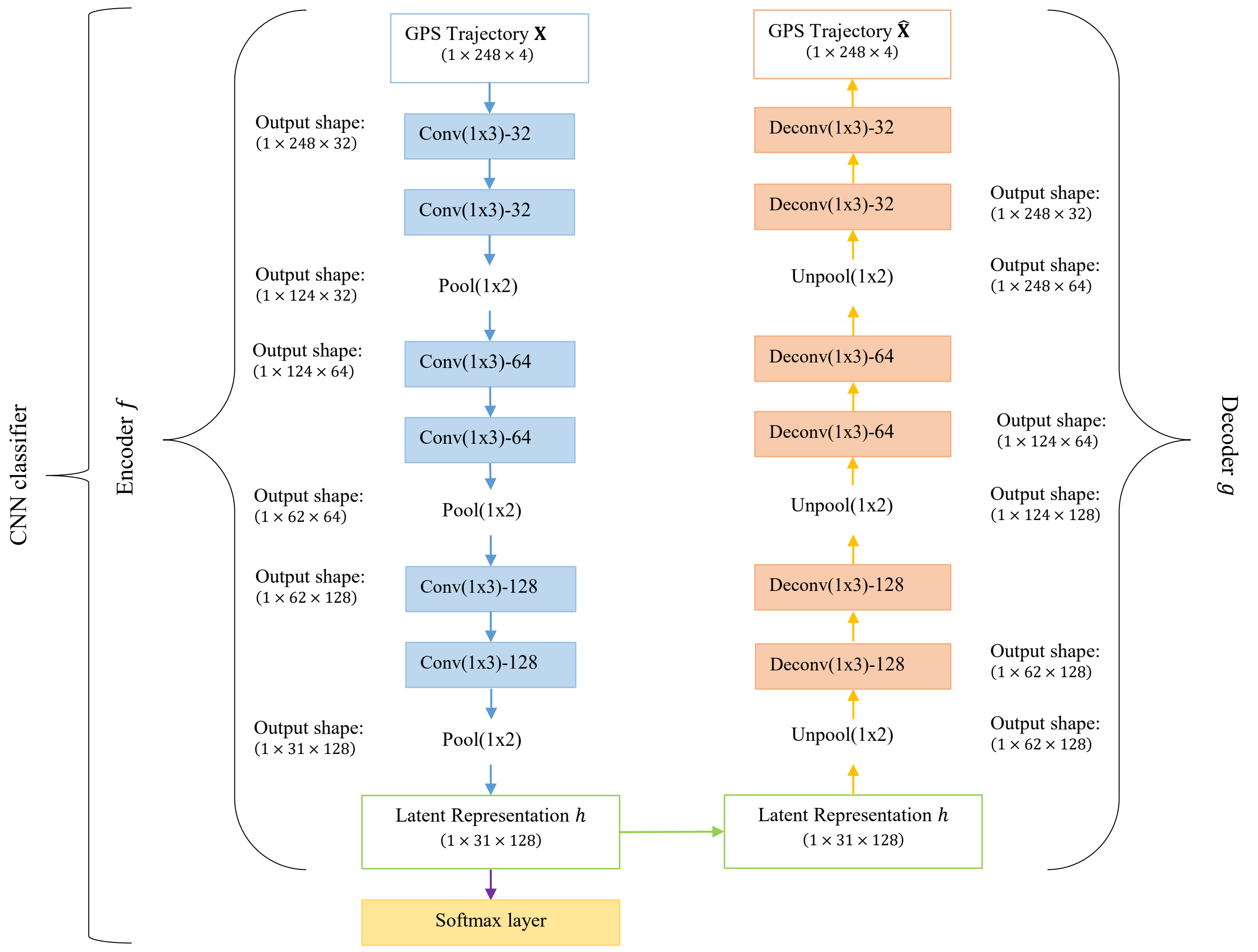
CNN classifier
Encoder f
GPS Trajectory $\mathbf{X}$ (1 × 248 × 4)
Output shape: (1 × 248 × 32)
Conv(1x3)-32
Conv(1x3)-32
Output shape: (1 × 124 × 32)
Pool(1x2)
Output shape: (1 × 124 × 64)
Conv(1x3)-64
Conv(1x3)-64
Output shape: (1 × 62 × 64)
Pool(1x2)
Output shape: (1 × 62 × 128)
Conv(1x3)-128
Conv(1x3)-128
Output shape: (1 × 31 × 128)
Pool(1x2)
Latent Representation h (1 × 31 × 128)
Softmax layer
GPS Trajectory $\hat{\mathbf{X}}$ (1 × 248 × 4)
Deconv(1x3)-32
Deconv(1x3)-32
Output shape: (1 × 248 × 32)
Unpool(1x2)
Output shape: (1 × 248 × 64)
Deconv(1x3)-64
Deconv(1x3)-64
Output shape: (1 × 124 × 64)
Unpool(1x2)
Output shape: (1 × 124 × 128)
Deconv(1x3)-128
Deconv(1x3)-128
Output shape: (1 × 62 × 128)
Unpool(1x2)
Output shape: (1 × 62 × 128)
Latent Representation h (1 × 31 × 128)
Decoder g

Fig. 3: The architecture of our semi-supervised framework, which consists of the convolutional-deconvolutional autoencoder and CNN classifier. The layers parameters are represented as "(filter size)-(number of filters)" for Conv. and Deconv. layers, and "(pooling size)" for pooling and unpooling layers. The "Output shape" denotes the output size of the corresponding layer, which is shown only when the output size changes.

two parts: (1) the encoder function that maps the input data into the latent representation $h = f(\mathbf{X})$, and (2) the decoder function that reconstructs the original data from the latent representation $\hat{\mathbf{X}} = g(h)$. The latent representation h often contains more useful properties than the original input data $\mathbf{X}$ (Goodfellow et al., 2016). In our framework, the functions f and g are deep convolutional and deconvolutional networks, respectively.

As shown in Fig. 3, the encoder function f consists of three sets of layers, where each set has two convolutional layers followed by a max pooling layer. The input to the decoder is $\mathbf{X}_{comb}$. Since the spatial size of $\mathbf{X}_{comb}$ is small in our application, a small filter size (1×3) is used for all convolutional layers while stride is equal to 1. Using smaller receptive fields leads to reducing the number of parameters and mitigating the overfitting problem [29]. The number of filters starts from 32 in the first set of convolutional layers, and then increase by a factor of 2 (i.e., 64 and 128 filters for the second and third sets, respectively). The padding setting of convolutional layers is configured so as to preserve the spatial dimension after each convolution operation. The filter size of the max-pooling layer is (1×2) with the stride 2. According to the mentioned settings, the spatial size reduces by a half size of the previous layer only after max-pooling layers and remains unchanged after convolutional layers. In order to introduce the nonlinearity in the model, the convolved neurons are activated by the Rectified Linear Unit ($ReLU$) function. Compared to other functions such as $tanh$ and $sigmoid$ functions, the learning rate of the CNN with $ReLU$ is much faster [30]. The tensor h is the output of the last layer in the encoder part. Note that we do not use a fully-connected layer as the last layer to force the latent representation h into a vector form. Based on our training observation, collapsing the latent representation h with 3-dimension into a 1-dimension vector deteriorates the framework performance. The tensor h in Fig. 3, for instance, has the shape size $h \in \mathbb{R}^{(1\times 31\times 128)}$.

The decoder function g has the same number of layers as the encoder. The input volume for each layer of the decoder is the output volume of the corresponding encoder layer in the encoder part. Thus, the decoder requires performing the inverse operation so as to generate an output with the same size of the input in the corresponding layer in the encoder part. Deconvolutional layer is the transposed convolutional layer, where transformation is going in the opposite direction of a normal convolution [31]. Unpooling is the conjugate layer of the pooling layer, which repeats the width of its input based on a user-defined factor. Since the pooling window size was set to (1×2), the unpooling size is also set to 2. For example, in Fig. 3, the tensor $h \in \mathbb{R}^{(1 \times 31 \times 128)}$ is first passed into the unpooling layer, which generates a tensor with the size $(1 \times 62 \times 128)$. Afterward, the output feature map from the unpooling layer is fed into a deconvolutional layer, which results in a tensor with the same shape $(1 \times 62 \times 128)$. Except for the last layer, the activation function for all deconvolutional layers is $ReLU$. Proceeding the same operations, the last deconvolutional layer produces an output with the same shape of the original input, denoted as $\hat{\mathbf{X}}_{comb} \in \mathbb{R}^{(1 \times M \times 4)}$. Since the input layer $\mathbf{X}_{comb}$ has been normalized into the range $[0, 1]$, the $sigmoid$ function is deployed as the activation function of the last deconvolutional layer. The $sigmoid$ is a nonlinear function that maps the values of $\hat{\mathbf{X}}_{comb}$ in between 0 to 1.

As $\mathbf{X}_{comb}$ and $\hat{\mathbf{X}}_{comb}$ are composed of continuous-valued features, we use the squared Euclidean distance as the loss function for the Conv-AE. Accordingly, the reconstruction error for every SE is computed as follows:

$$l^{Conv-AE} = \sum_i (\hat{x}_i - x_i)^2 \tag{7}$$

where $\hat{x}_i$ and x_i are the corresponding elements of the matrices $\hat{\mathbf{X}}_{comb}$ and $\mathbf{X}_{comb}$, respectively. The above error is averaged across the training batch in each iteration.

4.2.2 CNN Classifier

Our CNN classifier contains a stack of convolutional layers with one fully-connected layer. The convolutional part is exactly the same as the encoder function, yet receives $\mathbf{X}_l$ as the input layer. The output of the last convolutional layer is directly fed into a $softmax$ layer to generate a probability distribution over the transportation labels, denoted as $P_l = \{p_{l,1}, \cdots, p_{l,K}\}$, where K is the number of transportation modes. Thus, the CNN classifier can be viewed as a categorical logistic regression layer on the top of the encoder part, as depicted in Fig. 3. Note that using some fully connected layers between the last convolutional layer and the $softmax$ layer (i.e., deploying a multi-layer perceptron) does not improve the performance of our network. The common categorical cross-entropy is used as the loss function for the CNN classifier. The loss function for every labeled SE is formulated as follows:

$$l^{labeled-classifier} = -\sum_{i=1}^{K} y_{l,i} \log(p_{l,i}) \tag{8}$$

where $y_{l,i} \in \mathbf{Y}_l$ is a binary indicator if the class i is the true transportation label for the sample $\mathbf{X}_l$. $\mathbf{Y}_l$ is the true label for

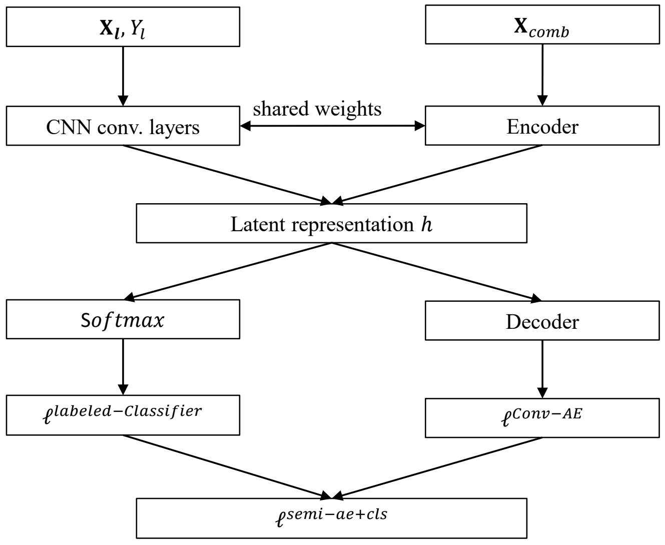

Fig. 4: Flow for jointly training the supervised and unsupervised components of the proposed semi-supervised architecture, depicted in Fig. 3

$\mathbf{X}_l$, encoded as a one-of-K vector. Analogous to the Conv-AE, the cross entropy loss is averaged across the training batch in each iteration.

4.2.3 Training Strategy

Our main training strategy is to simultaneously train the Conv-AE and CNN classifier, denoted as **Semi-AE+Cls**. The rationale behind this joint training strategy is to extract useful information from the underlying distribution of the input data through the Conv-AE, meanwhile enhancing the discrimination ability of the architecture using the classifier. As shown in Fig. 4, the encoder part of Conv-AE and the convolutional part of the CNN classifier, which have the same structure, need to share the same weights. As a consequence, in every weights update, the latent representation matrix h obtained by the encoder function is equivalent to the output of the last convolutional layer in the classifier. The unsupervised and supervised components of our proposed network are jointly learned by minimizing the following total loss function, which is a linear combination of Eqs. (7) and (8):

$$l^{semi-ae+cls} = \alpha l^{Conv-AE} + \beta l^{labeled-classifier} \tag{9}$$

where α and β are the model hyperparameters that make a balance between the relative importance of the two losses in Eqs. (7) and (8). The appropriate schedule for varying values of α and β over training epochs is an integral part of our model learning. Details for tuning these two hyperparameters are elaborated in the next section. Note that we use Adam optimizer as the optimization technique [32], Glorot uniform initializer for initializing the layers weights [22], and a dropout regularization with the dropout ratio 0.5 before the $softmax$ layer to overcome the overfitting problem [23].

4.2.4 Hyperparameters: α and β

Generally, two high-level tactics are applicable for scheduling α and β over training epochs: (1) Putting the initial focus on the unsupervised task and gradually shift the focus towards the supervised model. In other words, keep the value of α large and β small over the first epochs, and then slightly decrease α and increase β for next epochs. The key motivation behind this strategy is to first learn the high-level features of GPS trajectories and next refine the learned features to be more discriminating for the classification task [24], (2) The second schedule is the other way around by first assigning more weights to the supervised task and then increasing the effect of unsupervised learning. This scheme is expected to help the optimization process in two aspects: (a) to control the influence of unlabeled samples on the classification performance as the ultimate goal of the model, and (b) to avoid getting stuck in poor local minima points [33].

Our proposed training procedure for scheduling α and β over training epochs consists of two steps:

- **Setting $\boldsymbol{\alpha = 1}$ and $\boldsymbol{\beta = 1}$:** At the first step, both the Conv-AE and CNN classifier are simultaneously trained while they have the same weights. Training continues through several epochs until the validation score drops down. Training in this step is stopped after two epochs with no further improvement. Indeed, the main goal in the first step is to obtain the best possible performance without making any trade-off between reconstruction and cross-entropy errors. The weights with the best validation score are restored for the next step.
- **Setting $\boldsymbol{\alpha \in [1, 1.5]}$ and $\boldsymbol{\beta = 0.1}$:** No further improvement is achieved by the previous setting, which mainly stems from the overfitting problem and/or getting stuck in local minima. Dramatically reducing the effect of the supervised component can act like a sharp perturbation and take the optimization out of local minima. The unsupervised weight can be kept fixed or increased a bit. In our application, increasing $\boldsymbol{\alpha}$ up to 1.5 yields nearly the same performance. Continuing training with the new setting gives another chance to optimization so as to move towards better local minima. The same as the first step, the training is stopped when the validation score is not improved after two consecutive epochs. The optimal weights are restored for classifying the test set.

In addition to implementing two steps for training, our proposed schedule for varying balancing parameters differs from similar joint training strategies in two main aspects as follows:

- Neither fixed values nor annealing strategies are used over training epochs. In the annealing strategy, the value of α (or β) gradually decreases during the training to transit the focus towards the supervised (or unsupervised) task in the last training iterations [24].
- Unlike other studies [24], [25], the effect of supervised task, rather than unsupervised component, is significantly reduced in the last training iterations.
- Unlike other semi-supervised systems with joint training schemes [25], [24], [34], balancing parameters are considered for both unsupervised and supervised components.

5 Experimental Results

In this section, we will evaluate the performance of our proposed deep semi-supervised network on the GeoLife GPS trajectory dataset. First the GeoLife dataset and settings used for preparing data are described. Then, several supervised and semi-supervised baselines for comparison against our model are introduced. The prediction performance of our model is evaluated by comparing to baselines based on common classification metrics. The proposed training strategy and the image-like input layer are also assessed against several alternatives.

5.1 Dataset Description

The proposed model is examined and validated on the GPS trajectories collected by 182 users in the GeoLife project [35]. 69 users have labeled their trajectories with transportation modes while the remaining users left their trajectories unlabeled. Since the trajectory and label files of the 69 users are in separate, an efficient matching process for labeling GPS segments is implemented. It is worth noticing that not all trajectories of those 69 users have been annotated, which in turn are added to the unlabeled data. Although many kinds of transport modes have been labeled by users, only transport modes that constitute the significant portion of the dataset are considered in our analysis. In accordance with the user guideline linked to the published dataset [35], driving is assigned as the label of both taxi and car. Also, all reported rail-based modes (i.e., subway, train, and railway) is assigned to the train mode due to the layout of the rail-based system explained in the user guide. Therefore, our final list of transportations modes is Y = {walk, bike, bus, driving, train}. The time-interval threshold for dividing a user's GPS trajectory T into trips is set to 20 minutes [5]. The minimum number of GPS points, the maximum number of GPS points (i.e., M), the minimum distance, and the minimum duration of a GPS SE sample are set to 20, 248, 150 meters, and 1 minute. The maximum allowable speed and acceleration pertaining to each mode are provided in Table 2, which have been defined using several reliable online sources and the engineering justification (e.g., existing speed limits, current vehicle and humans power).

After applying the described procedure in Section 4.1 to the labeled and unlabeled GPS trajectories using the above-mentioned settings, the distribution of the created labeled samples among various modes and the number of unlabeled samples are illustrated in Table 2.

5.2 Baseline Methods

We compare our model with two sets of baselines: (1) Supervised algorithms, and (2) Semi-supervised algorithms. With regard to the first group, widely used traditional supervised algorithms in the literature of transport mode detection are deployed for comparison, including K-Nearest Neighborhood (KNN), RBF-based Support Vector Machine (SVM), Decision Tree (DT), and Multilayer Perceptron (MLP). Hand-crafted features described in Section 3.3 are used as features for the classification task, as the most acceptable manual GPS trajectories' attributes in the literature. Our CNN classifier with the same settings as in the proposed

TABLE 2: Number of Labeled GPS SE for each Transportation Mode, Number of Unlabeled GPS SE, as well as the Maximum Speed and Acceleration Associated with each Transportation Mode. NA: Not Applicable.

Mode	**No. of SE**	**Max. $S(m/s)$**	**Max. $A(m/s^2)$**
Walk	6640	7	3
Bike	3808	12	3
Bus	6051	34	2
Driving	4323	50	10
Train	3287	34	3
Total Labeled	24109	NA	NA
Unlabeled	72506	NA	NA

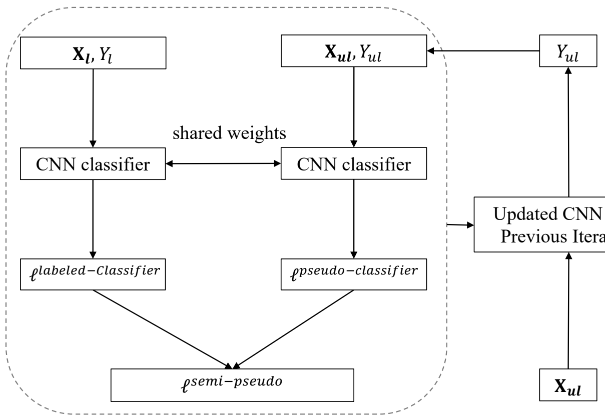
$\mathbf{X}_l, Y_l$
$\mathbf{X}_{ul}, Y_{ul}$
Y_{ul}
shared weights
CNN classifier
CNN classifier
Updated CNN from Previous Iteration
$\ell^{labeled-Classifier}$
$\ell^{pseudo-classifier}$
$\ell^{semi-pseudo}$
$\mathbf{X}_{ul}$

Fig. 5: Pseudo-label training strategy.

model is used as another baseline, whereas trained in a purely supervised manner on the labeled image-like GPS segments.

With regard to semi-supervised algorithms, we train our semi-supervised structure with three more distinct training strategies, described as below. According to categorization of training strategies described in Section 2, the first two methods are grouped as two-step training strategies while the last one is another type of joint training strategy.

- **Semi-Two-Steps:** First, the Conv-AE is trained on both labeled and unlabeled trajectories. Then, the labeled data are transformed to the latent representation using the encoder part. In the second step, the transformed data are trained on a standard supervised algorithm, which is a logistic regression (i.e., the $softmax$ layer) in our case. The primary motivation for this approach is to first extract abstract features from GPS trajectories, then train a supervised learning algorithm on the top of these efficient features rather than the original features. The loss functions for the Conv-AE and logistic regression are Eqs. (7) and (8), respectively.
- **Semi-Layer-Wise:** Except the $softmax$ layer, every layer in the CNN classifier is sequentially pre-trained from top to bottom on labeled and unlabeled data using one-hidden-layer autoencoder. The initial weights of the CNN classifier, obtained through the pre-training phase, are then fine-tuned on the labeled set in a supervised fashion. Thus, the optimizer needs to perform a local search since the network has already obtained a good initial set of weights through the pre-training phase [18]. The loss functions for every one-hidden-layer autoencoder and CNN classifier are Eqs. (7) and (8), respectively.
- **Semi-Pseudo-Label**: As shown in Fig. 5, two CNN classifiers with the same structures and shared layers are simultaneously trained in a supervised fashion, one on labeled and the other on unlabeled data. Pseudo label, Y_{ul}, is the predicted probability distribution over labels for an unlabeled sample $\mathbf{X}_{ul}$, using the updated weights from the previous training iteration. The overall loss function for this strategy is defined as follows:

$$l^{semi-pseudo} = \alpha l^{pseudo-classifier} + \beta l^{labeled-classifier}$$
$$l^{pseudo-classifier} = -\sum_{i=1}^{K} y_{ul,i} \log(p_{ul,i}) \quad (10)$$

where $y_{ul,i} \in Y_{ul}$ and $p_{ul,i} \in P_{ul}$ are the predicted probability for the class i based on the updated weights in the previous and current training iteration, respectively. Analogous to **Semi-AE+Cls**, α and β are the balancing parameters. Neglecting hyperparameters, Eq. (10) is equivalent to Eq. (8) with an Entropy Regularization on the class probabilities [33]. Minimizing such a regularization reduces the overlap of the labels probability distribution, which in turn forces the decision boundary to lie in low-density regions and improve the generalization performance [26].

5.3 Performance Evaluation

The performance quality of our proposed model is measured using two common classification metrics:

- *Accuracy* - it is computed as the fraction of samples in the test set that are correctly classified.
- *F-measure* - it is defined as the harmonic average of the precision and recall in the test set, where the higher value indicates both precision and recall are reasonably high.

$$F-measure = \frac{2\times Precision\times Recall}{Precision+Recall}$$

In all experiments, models are trained and tested using stratified 5-fold cross-validation and average values (along with the standard deviations) of the results on all 5-folds are reported. Note that the stratified 5-fold cross-validation is only applied to labeled data while the whole unlabeled data are used for training in semi-supervised models. Using stratified sampling, 10% of the training-labeled data in each fold is selected as the validation set for the early-stopping method in deep-learning models. All described data processing and models are implemented in Python programming environment with the help of TensorFlow for deep learning models and scikit-learn for classical supervised algorithms. All experiments are run on a computer with a single GPU. The source codes related to all data processing and models utilized in this study are available at https://github.com/sinadabiri/Deep-Semi-Supervised-GPS-Transport-Mode

5.4 Results and Discussion

Tables 3 and 4 provide the performance results of our proposed model and baselines in terms of accuracy and F-measure, respectively. Every model is trained for various amounts of labeled data so as to investigate the effectiveness of the models when a different number of labeled data are available.

What stands out in Table 3 is the superiority of our proposed model and training strategy in comparison with baselines. Except for 10% labeled data that DT works better, our semi-supervised model consistently outperforms other methods for all percentages of labeled data. With respect to supervised algorithms, it is apparent that only CNN and DT are competitive as the test accuracy for other traditional learning methods are considerably low. In comparison between supervised and semi-supervised algorithms, the results indicate that semi-supervised algorithms perform better than supervised techniques. Indeed, modeling the distribution of input data through unsupervised learning techniques with the help of unlabeled data can potentially ameliorate the generalization ability of supervised task. Focusing on only semi-supervised algorithms, it is obvious that the prediction quality of joint training strategies (i.e., Semi-AE+Cls and Semi-Pseudo Label) are significantly higher than the two-step schemes (i.e., Semi-Two-Steps and Semi-Layer-Wise). Such evidence confirms that, with aid of efficient initialization techniques, concurrent preserving the reconstruction and classification abilities yields a better performance compared to disjoint training approaches. As an overall comparison, our model achieves on average 6.2% higher accuracy compared to other methods over different amounts labeled data, excluding the low-quality supervised algorithms (i.e., KNN, SVM, and MLP).

Achieving high accuracy is only a positive starting point for having a reliable classifier. An effective and unambiguous way for evaluating the performance of a classifier is to use other measures such as precision, recall, and F-measure. As can be seen in Table 4, F-measure for all models is almost the same as test accuracy, which evidently confirms our findings and reasoning based on results in Table 3.

Turning now to our proposed schedule for tuning balancing parameters α and β, Table 5 presents the test accuracy of our semi-supervised model (i.e., semi-AE+Cls), according to several schedules for tuning hyperparameters in the loss function. In the schedule #1, the hyperparameter α gradually decreases from 1 to the minimum value 0.1 while the hyperparameter β is fixed to 1 during training. In fact, this schedule gradually shifts the focus solely on the supervised component over the training iterations. Schedule #2 is analogous to the schedule #1, yet with keeping the focus on the unsupervised task. Schedule #3 maintains the balancing parameters equal to 1 during the whole training process. Note that the schedules (1-3) have only one stage and the training is stopped when no further improvement is achieved after two consecutive epochs. Schedules #4 and #5 have two stages. In the first stage, only one component (i.e., either supervised or unsupervised component) is trained until the early stopping criterion terminates training. Then, in the second stage, the training continues by reversing the focus to the part that has not been trained in the first stage. Training at this stage is stopped until no further improvement is achieved after two consecutive epochs. From Table 5, it can be seen that our proposed schedule for tuning balancing parameters improves the model performance more than other alternatives. The results reveal that deploying an effective tuning schedule can simply increase the model accuracy. It is worth noticing that the schedules #1-#5 are the best examples with the highest accuracy among many other possible schedules.

Furthermore, the performance measures of Semi-Pseudo Label in Tables 3 and 4 supports the effectiveness of our proposed hyperparameter schedule. Note that Semi-Pseudo Label is the most competitive technique to ours and is jointly trained by varying its balancing parameters according to our proposed schedule.

The quality of our proposed image-like GPS tensor in Fig. 2 is examined by tracking the effectiveness of various motion-feature combinations. Table 6 summarizes the performance metrics when the input layer is created based on a single or a combination of features. Features are defined based on Eqs. (1-6). The number of motion features determines the number of channels in the image-like tensor. For example, an image-like tensor with only S information is constructed based on the described procedure in Section 4.1, yet with only one channel rather than four. Reported values in Table 6 are the average of 5-fold cross-validation when all labeled data are available. As can be observed, speed (A), relative distance (RD), and acceleration (A) are the most effective features in a stand-alone operation. However, a combination of features leads to obtaining higher accuracy and F-measure. Among several potential combinations, our image-like configuration, that fuses relative distance, speed, acceleration, and jerk, attains the best performance. Results also reveal that using all types of motion characteristics (i.e., a 6-channel image-like) does not necessarily result in the highest score, which calls for an appropriate selection of features. Another interesting finding is that the proposed image-like configuration even outperforms the combination of top single features (i.e., $RD + S + A + \Delta t$).

6 Conclusion

We propose a semi-supervised convolutional framework for identifying transportation modes from GPS trajectory data. The network consists of a convolutional-deconvolutional autoencoder and a CNN classifier as the unsupervised and supervised components, respectively. First, a raw GPS trajectory was converted into an efficient 4-channel image-like tensor so as to utilize the convolutional operation for automatically extracting features rather than using hand-crafted features. As a key objective of this study, unlabeled trajectories were also exploited and fed into the proposed network in order to improve the model inference accuracy. Both supervised and unsupervised components of the architecture were simultaneously trained while an optimal schedule was deployed for varying the balancing parameters between reconstruction and classification losses. Our extensive experiments demonstrated the superiority of our proposed image-like GPS tensor, semi-supervised model, and hyperparameter-tuning schedule. Furthermore, comparing the inference performance of semi-supervised

TABLE 3: Comparison of Accuracy values for different Supervised and Semi-supervised models with varying sizes of labeled data.

	Proportion of labeled data in the training data				
Model	**10%**	**25%**	**50%**	**75%**	**100%**
Supervised-KNN	0.469 (±0.015)	0.508 (±0.012)	0.549 (±0.014)	0.567 (±0.015)	0.579 (±0.015)
Supervised-SVM	0.417 (±0.006)	0.460 (±0.005)	0.470 (±0.006)	0.517 (±0.009)	0.532 (±0.010)
Supervised-DT	**0.661 (±0.014)**	0.672 (±0.013)	0.678 (±0.017)	0.689 (±0.012)	0.694 (±0.014)
Supervised-MLP	0.274 (±0.093)	0.309 (±0.107)	0.331 (±0.036)	0.347 (±0.069)	0.354 (±0.084)
Supervised-CNN	0.568 (±0.044)	0.617 (±0.042)	0.687 (±0.021)	0.719 (±0.019)	0.741 (±0.024)
Semi-Two-Steps	0.544 (±0.019)	0.562 (±0.035)	0.588 (±0.016)	0.600 (±0.012)	0.605 (±0.015)
Semi-Layer-Wise	0.577 (±0.028)	0.608 (±0.021)	0.676 (±0.025)	0.690 (±0.023)	0.694 (±0.033)
Semi-Pseudo-Label	0.589 (±0.020)	0.663 (±0.023)	0.707 (±0.021)	0.733 (±0.021)	0.754 (±0.018)
Semi-AE+Cls (ours)	0.629 (±0.010)	**0.693 (±0.019)**	**0.732 (±0.017)**	**0.750 (±0.016)**	**0.768 (±0.016)**

TABLE 4: Comparison of F-measure for Various Supervised and Semi-Supervised Models with Different Sizes of Labeled Data

	Proportion of labeled data in the training set				
Model	**10%**	**25%**	**50%**	**75%**	**100%**
Supervised-KNN	0.440 (±0.017)	0.488 (±0.017)	0.531 (±0.017)	0.551 (±0.017)	0.564 (±0.017)
Supervised-SVM	0.337 (±0.003)	0.385 (±0.008)	0.429 (±0.013)	0.455 (±0.017)	0.476 (±0.018)
Supervised-DT	**0.662 (±0.014)**	0.672 (±0.014)	0.678 (±0.017)	0.689 (±0.012)	0.695 (±0.014)
Supervised-MLP	0.194 (±0.090)	0.227 (±0.129)	0.269 (±0.043)	0.252 (±0.064)	0.266 (±0.094)
Supervised-CNN	0.533 (±0.063)	0.560 (±0.044)	0.678 (±0.022)	0.710 (±0.020)	0.734 (±0.026)
Semi-Two-Steps	0.512 (±0.026)	0.541 (±0.038)	0.574 (±0.016)	0.584 (±0.014)	0.589 (±0.019)
Semi-Layer-Wise	0.553 (±0.031)	0.590 (±0.023)	0.663 (±0.026)	0.678 (±0.024)	0.681 (±0.035)
Semi-Pseudo-Label	0.582 (±0.020)	0.654 (±0.025)	0.701 (±0.022)	0.728 (±0.021)	0.749 (±0.019)
Semi-AE+Cls (ours)	0.615 (±0.011)	**0.683 (±0.019)**	**0.725 (±0.017)**	**0.745 (±0.017)**	**0.764 (±0.017)**

models with supervised techniques revealed that gleaning more information about the distribution of input data from unlabeled data provides an opportunity to improve the prediction quality and generalization ability of the network. Nonetheless, only deployment of semi-supervised models is not a guarantee for obtaining higher accuracy compared to supervised algorithms. Thus, selecting an appropriate training strategy is essential for enhancing the ultimate classification performance in semi-supervised architectures.

Acknowledgements

This work was supported in part by the National Science Foundation grants IIS-1619028, IIS-1707498 and IIS-1646881.

REFERENCES

[1] L. Wu, B. Yang, and P. Jing, "Travel mode detection based on gps raw data collected by smartphones: a systematic review of the existing methodologies," *Information*, vol. 7, no. 4, p. 67, 2016.

[2] S. Dabiri and K. Heaslip, "Transport-domain applications of widely used data sources in the smart transportation: A survey," *arXiv preprint arXiv:1803.10902*, 2018.

[3] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, "Transportation mode detection using mobile phones and gis information," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 54–63, ACM, 2011.

[4] T. Feng and H. J. Timmermans, "Transportation mode recognition using gps and accelerometer data," *Transportation Research Part C: Emerging Technologies*, vol. 37, pp. 118–130, 2013.

[5] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceedings of the 17th international conference on World Wide Web*, pp. 247–256, ACM, 2008.

[6] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Proceedings of the 10th international conference on Ubiquitous computing*, pp. 312–321, ACM, 2008.

[7] Z. Xiao, Y. Wang, K. Fu, and F. Wu, "Identifying different transportation modes from trajectory data using tree-based ensemble classifiers," *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, p. 57, 2017.

[8] Z. Sun and X. J. Ban, "Vehicle classification using gps data," *Transportation Research Part C: Emerging Technologies*, vol. 37, pp. 102–117, 2013.

[9] H. Menp, L. Andrei, and L. M. L. Jose, "Travel mode estimation for multi-modal journey planner," *Transportation Research Part C: Emerging Technologies*, vol. 82, pp. 273–289, 2017.

[10] S. Dabiri and K. Heaslip, "Inferring transportation modes from gps trajectories using a convolutional neural network," *Transportation Research Part C: Emerging Technologies*, vol. 86, pp. 360–371, 2018.

[11] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

[12] T. Nick, E. Coersmeier, J. Geldmacher, and J. Goetze, "Classifying means of transportation using mobile sensor data," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–6, IEEE, 2010.

[13] M. Rezaie, Z. Patterson, J. Y. Yu, and A. Yazdizadeh, "Semi-supervised travel mode detection from smartphone data," in *Smart Cities Conference (ISC2), 2017 International*, pp. 1–8, IEEE, 2017.

[14] Y. Endo, T. Hiroyuki, N. Kyosuke, and K. Akihisa, "Deep feature extraction from trajectories for transportation mode estimation,"

TABLE 5: Comparison of Accuracy for Various Hyperparameter Schedules Along with Different Sizes of Labeled Data

	Schedule for α and β		**Proportion of labeled data in the training set**				
#	**stage 1**	**stage 2**	**10%**	**25%**	**50%**	**75%**	**100%**
1	$\alpha : 1 \rightarrow 0.1$ $\beta : 1$	NA	0.603 (±0.018)	0.675 (±0.015)	0.709 (±0.008)	0.712 (±0.020)	0.747 (±0.022)
2	$\alpha : 1$ $\beta : 1 \rightarrow 0.1$	NA	0.618 (±0.017)	0.670 (±0.024)	0.715 (±0.015)	0.721 (±0.017)	0.741 (±0.012)
3	$\alpha : 1$ $\beta : 1$	NA	0.625 (±0.018)	0.667 (±0.016)	0.716 (±0.015)	0.734 (±0.019)	0.745 (±0.022)
4	$\alpha : 1$ $\beta : 0$	$\alpha : 1$ $\beta : 1$	0.551 (±0.021)	0.564 (±0.214)	0.703 (±0.028)	0.715 (±0.023)	0.739 (±0.024)
5	$\alpha : 0$ $\beta : 1$	$\alpha : 1$ $\beta : 1$	0.590 (±0.023)	0.671 (±0.022)	0.711 (±0.025)	0.732 (±0.023)	0.755 (±0.022)
6 (ours)	$\alpha : 1$ $\beta : 1$	$\alpha : 1$ $\beta : 0.1$	**0.629 (±0.010)**	**0.693 (±0.019)**	**0.732 (±0.017)**	**0.750 (±0.016)**	**0.768 (±0.016)**

TABLE 6: Comparison of Accuracy and F-measure for Several Configurations of the Image-like Tensor Depicted in Fig. 2

Single Feature	**Accuracy**	**F-measure**	**Feature Combination**	**Accuracy**	**F-measure**
RD	0.529 (±0.119)	0.492 (±0.119)	$RD + S$	0.702 (±0.022)	0.691 (±0.023)
Δt	0.375 (±0.054)	0.352 (±0.087)	$RD + S + A$	0.763 (±0.016)	0.758 (±0.016)
S	0.702 (±0.021)	0.691 (±0.022)	$RD + \Delta t + S + A$	0.755 (±0.018)	0.750 (±0.022)
A	0.481 (±0.169)	0.415 (±0.022)	$RD + \Delta t + S + A + J + BR$	0.752 (±0.026)	0.748 (±0.026)
J	0.275 (±0.000)	0.119 (±0.000)	$RD + S + A + BR$	0.752 (±0.018)	0.741 (±0.018)
BR	0.295 (±0.019)	0.178 (±0.056)	$RD + S + A + J$	**0.768 (±0.016)**	**0.764 (±0.017)**

in *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 54–66, 2016.

[15] H. Wang, L. GaoJun, D. Jianyong, and Z. Lei, "Detecting transportation modes using deep neural network," in *IEICE TRANSACTIONS on Information and Systems*, pp. 1132–1135, 2017.

[16] R. Johnson and T. Zhang, "Supervised and semi-supervised text categorization using lstm for region embeddings," *arXiv preprint arXiv:1602.02373*, 2016.

[17] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, "Semi-supervised learning with ladder networks," in *Advances in Neural Information Processing Systems*, pp. 3546–3554, 2015.

[18] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, pp. 153–160, 2007.

[19] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[20] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems*, pp. 3079–3087, 2015.

[21] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial training methods for semi-supervised text classification," *arXiv preprint arXiv:1605.07725*, 2016.

[22] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] Y. Zhang, S. Dinghan, W. Guoyin, G. Zhe, H. Ricardo, and C. Lawrence, "Deconvolutional paragraph representation learning," in *Advances in Neural Information Processing Systems*, pp. 4172–4182, 2017.

[25] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the conference on empirical methods in natural language processing*, pp. 151–161, Association for Computational Linguistics, 2011.

[26] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on Challenges in Representation Learning, ICML*, vol. 3, p. 2, 2013.

[27] T. Vincenty, "Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations," *Survey review*, vol. 23, no. 176, pp. 88–93, 1975.

[28] O. Bagdadi and A. Várhelyi, "Development of a method for detecting jerks in safety critical events," *Accident Analysis & Prevention*, vol. 50, pp. 83–91, 2013.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[31] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535, IEEE, 2010.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in *Advances in neural information processing systems*, pp. 529–536, 2005.

[34] M. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *Proceedings of the 25th international conference on Machine learning*, pp. 792–799, ACM, 2008.

[35] Y. Zheng, F. Hao, X. Xing, M. Wei-Ying, and L. Quannan, "Geolife gps trajectory dataset-user guide," 2011.

Sina Dabiri received his MS in Transportation Engineering from Sharif University of Technology, Tehran, Iran in 2012. Currently, he is simultaneously pursuing his Ph.D. in the Department of Civil Engineering and MS in the Department of Computer Science at Virginia Tech. His research interests include machine learning and deep learning in intelligent transportation systems (ITS).

Chang-Tien Lu received the MS degree in computer science from the Georgia Institute of Technology in 1996 and the PhD degree in computer science from the University of Minnesota in 2001. He is an associate professor in the Department of Computer Science, Virginia Tech. He served as general co-chair of the 20th IEEE International Conference on Tools with Artificial Intelligence in 2008 and 17th ACM International Conference on Advances in Geographic Information Systems in 2009. He is also serving as vice chair of the ACM Special Interest Group on Spatial Information (ACM SIGSPATIAL). His research interests include spatial databases, data mining, geographic information systems, and intelligent transportation systems.

Kevin Heaslip is an Associate Professor of Civil Engineering at Virginia Tech. He earned a B.S. and M.S. from Virginia Tech, and a Ph.D. from the University of Massachusetts Amherst. He also serves as the Associate Director of the Information Systems Laboratory at the Hume Center for National Security and Technology. His research interests lie in the connections of transportation, smart cities, resilience, and cybersecurity. He is a registered professional engineer in New Hampshire with more than 15 years' experience in the transportation fields. He has also been a Principal or Co-Principal Investigator of projects funded at over 18 million dollars.

Chandan K. Reddy is an Associate Professor in the Department of Computer Science at Virginia Tech. He received his Ph.D. from Cornell University and M.S. from Michigan State University. His primary research interests are Data Mining and Machine Learning with applications to Healthcare Analytics and Social Network Analysis. His research is funded by the National Science Foundation, the National Institutes of Health, the Department of Transportation, and the Susan G. Komen for the Cure Foundation. He has published over 95 peer-reviewed articles in leading conferences and journals. He received several awards for his research work including the Best Application Paper Award at ACM SIGKDD conference in 2010, Best Poster Award at IEEE VAST conference in 2014, Best Student Paper Award at IEEE ICDM conference in 2016, and was a finalist of the INFORMS Franz Edelman Award Competition in 2011. He is an associate editor of the ACM Transactions on Knowledge Discovery and Data Mining and PC Co-Chair of ASONAM 2018. He is a senior member of the IEEE and life member of the ACM.