Application of Deep Learning in Intelligent Transportation Systems

Sina Dabiri

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Civil Engineering

Kevin P. Heaslip, Chair
Montasir M. Abbas
Ryan M. Gerdes
Naren Ramakrishnan

December 12, 2018
Blacksburg, Virginia

Keywords: Deep Learning, Intelligent Transportation Systems, GPS Data, Twitter Data, Travel Mode Detection, Vehicle Classification, Traffic Information System, Machine Learning, Natural Language Processing

Application of Deep Learning in Intelligent Transportation Systems

Sina Dabiri

(ABSTRACT)

The rapid growth of population and the permanent increase in the number of vehicles engender several issues in transportation systems, which in turn call for an intelligent and cost-effective approach to resolve the problems in an efficient manner. A cost-effective approach for improving and optimizing transportation-related problems is to unlock hidden knowledge in ever-increasing spatiotemporal and crowdsourced information collected from various sources such as mobile phone sensors (e.g., GPS sensors) and social media networks (e.g., Twitter). Data mining and machine learning techniques are the major tools for analyzing the collected data and extracting useful knowledge on traffic conditions and mobility behaviors. Deep learning is an advanced branch of machine learning that has enjoyed a lot of success in computer vision and natural language processing fields in recent years. However, deep learning techniques have been applied to only a small number of transportation applications such as traffic flow and speed prediction. Accordingly, my main objective in this dissertation is to develop state-of-the-art deep learning architectures for resolving the transport-related applications that have not been treated by deep learning architectures in much detail, including (1) travel mode detection, (2) vehicle classification, and (3) traffic information system. To this end, an efficient representation for spatiotemporal and crowdsourced data (e.g., GPS trajectories) is also required to be designed in such a way that not only be adaptable with deep learning architectures but also contains efficient information for solving the task-at-hand. Furthermore, since the good performance of a deep learning algorithm is primarily contingent on access to a large volume of training samples, efficient data collection and labeling strategies are developed for different data types and applications. Finally, the performance of the proposed representations and models are evaluated by comparing to several state-of-the-art techniques in literature. The experimental results clearly and consistently demonstrate the superiority of the proposed deep-learning based framework for each application.

Application of Deep Learning in Intelligent Transportation Systems

Sina Dabiri

(GENERAL AUDIENCE ABSTRACT)

The rapid growth of population and the permanent increase in the number of vehicles engender several issues in transportation systems, which in turn call for an intelligent and cost-effective approach to resolve the problems in an efficient manner. Furthermore, the recent advances in positioning tools (e.g., GPS sensors) and ever-popularity of social media networks have enabled generation of massive spatiotemporal and crowdsourced data. This dissertation aims to leverage the advances in artificial intelligence so as to unlock the rick knowledge in the recorded data and in turn, optimizing the transportation systems in a cost-effective way. In particular, this dissertation seeks for proposing end-to-end frameworks based on deep learning models, as an advanced branch of artificial intelligence, as well as spatiotemporal and crowdsourced datasets (e.g., GPS trajectory and social media) for improving three transportation problems. *(1) Travel Mode Detection*, which is defined as identifying users' transportation mode(s) (e.g., walk, bike, bus, car, and train) when traveling around the traffic network. *(2) Vehicle Classification*, which is defined as identifying the vehicle's type (e.g., passenger car and truck) while moving in a traffic network. *(3) traffic information system based on social media networks*, which is defined as detecting traffic events (e.g., crash) and capturing traffic information (e.g., traffic congestion) on a real-time basis from users' tweets. The experimental results clearly and consistently demonstrate the superiority of the proposed deep-learning based framework for each application.

Dedication

*To my **fiancée, Sanam,** for your kindness, devotion, and endless support along the way. This humble work is a sign of my love to you!*

Acknowledgments

I would like to thank my great adviser, Dr. Kevin Heaslip, for his excellent cooperation and all of the opportunities I was given to conduct my research. I thank you very much for your endless support and guidance throughout my studies at Virginia Tech. It has been my utmost privilege to work with you.

In addition, I am very thankful to my committee members, Dr. Montasir Abbas, Dr. Ryan Gerdes, and Dr. Ramakrishnan, for their time and cooperation in the completion of this dissertation.

Finally, I would like to dedicate a special thanks to my parents and my sisters for their endless support and devotion although they were thousands of miles away from me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A land transportation network is any platform that permits people to move from an origin to a destination with various modes such as car, bus, train, walking, bicycle, etc. Despite the mobility and access benefits of transportation networks, cities have always been struggling with transportation-related issues (e.g., traffic congestion and air pollution) due to the ever-growing increase in the population and the number of vehicles. Various attitudes exist to resolve the above and other transportation-related issues. One way is to enforce laws on users. For example, preventing the private vehicle owners from entering in the central business district from morning until late afternoon leads to reducing congestion and forcing people to use the public transit. Augmenting the system capacity by constructing new infrastructures, such as bridges, tunnels, and increasing the number of lanes in expressways is another solution for reducing the congestion. However, a more cost-effective approach is to optimize the transportation network by extracting hidden knowledge in the data collected from various sources such as mobile phone networks, commuting smart cards, and traffic sensors. Fortunately, recent advances in sensing technologies, data management systems, distributed computing infrastructures, and artificial intelligence (AI) models have helped to address the transportation challenges more intelligently. Intelligent Transportation Systems (ITS) is a framework that integrates such technologies to ease congestion, minimize environmental impacts, and in general improve transportation safety, workability, and sustainability. ITS functional areas include advanced transportation management systems, advanced travel information systems, advanced vehicle control systems, commercial vehicle operations, advanced public transportation systems, and advanced rural transportation systems [81].

The general framework of ITS, as the key application of urban computing, contains four layers: 1) data collection 2) data management, 2) data analytics, and 3) service providing [100]. In the first step, transportation-related data are collected with the aid of smart devices located throughout the city and sensing technologies including traffic flow sensors, Global Positioning Systems (GPS), mobile phone cellular networks, Bluetooth, location-based social networks, transit smart cards, and environmental data [18]. In the second step, the heterogeneous data from various sources are organized using data preprocessing and data management techniques to prepare spatiotemporal and text information as the main inputs for the data analytics step. After collecting and providing the clean and validated data, AI techniques are employed to extract useful knowledge on traffic conditions and mobility behaviors. Such knowledge allows people and machines to act and decide more intelligently. Analyzing behavior of vehicles, developing real-time traveler information systems, and detecting the location of traffic incidents are only a few achievements yielded by rigorously mining the sensed data. In the service providing step, the output of analytic models (i.e., information on transport system status) are sent to users to select appropriate options according to their needs such as choosing the shortest route and the low-cost mode. Meanwhile, authorities are well informed of current shortcomings of systems and efficient strategies for outperforming the transportation network.

In accordance with the ITS framework, data preprocessing and analytical techniques significantly affect the quality of provided services in ITS. In this dissertation, I envision to deploy an advanced branch of AI, deep learning, in several transport-related applications. Although extensive research has been carried out on deep learning in a variety of computer science fields such as computer vision and natural language processing, researcher have not treated deep learning in transportation domains in much detail. Thus, my main objective in this dissertation is to leverage advances in deep learning architectures for resolving transport-related applications. To this end, the major prerequisite is to design an efficient representation for spatiotemporal data so as to not only be adaptable with deep learning architectures but also contains efficient information for solving the task-at-hand. Furthermore, since a good performance of deep learning architectures is primarily contingent on access to a large volume of training samples, efficient data collection and labeling strategies are required to be developed, depending on the data type

and the transport application. Accordingly, this dissertation seeks for answering the following research questions:

- What are the appropriate strategies for collecting, labeling, and augmenting various types of datasets in order to provide a large volume of training input samples and in turn, meet the core requirement of deep learning architectures?
- What are efficient representations for raw spatiotemporal and crowdsourced data (e.g., GPS trajectories) before passing them to deep learning architectures?
- To what extent can deep learning architectures bring novelty and improvement in the common transportation applications compared to traditional approaches?

In the following sections, first, the core concepts of deep learning architectures are introduced. Then, I expand how this category of AI have been deployed in the literature of ITS so far. Finally, my contributions for improving ITS applications with the aid of deep learning will be reviewed.

1.2 Deep learning

Machine learning (ML) is a field of AI that utilizes statistical techniques to learn hidden patterns from available data and make decisions on unseen records. The core task of a machine learner is to first build a general model on the probability distribution of training examples, and then generalize its experience on unseen examples [56]. The process of learning is highly dependent on the quality of data representation. A sample in a data set is represented by several pieces of information, called features. A classical ML technique seeks to learn how samples are correlated with their outcomes (e.g., various labels) according to those pre-defined features. However, the ML method is unable to affect the features designation. Thus, the performance of an ML technique for learning a task heavily depends on how well the features have already been designed [33].

Unfortunately, it might be difficult to extract efficient features for some tasks. In the vehicle-recognition task from images, for example, a car can be described

by the presence of some components such as wheels and windshields. At first glance, it might be simple to recognize an object with these components as a car; however, extracting these components from pixels of an image requires the computer algorithm to be robust against variation in size and shape, environmental changes such as day-time, illumination conditions, and occlusion due to shadowing [33]. Furthermore, the original content of a dataset may not be in an acceptable format for ML techniques. For example, a text document comprises a sequence of symbols, whereas a majority of ML techniques requires a fixed number of numerical features not sequences of words with variable lengths. Although a text document can be converted to a numerical vector by counting the frequency of each word occurrence in the text (a.k.a. bag-of-words representation), such an approach completely ignores the relative-position information of the words in the document.

A practical solution to the above-mentioned issues is to deploy representation learning, a branch of the ML field. The core objective of representation learning, a.k.a. feature learning, is to transform the raw input into a new set of features that contain more useful properties for the task-at-hand. Extracted features from representation learning is often supposed to result in the better performance of ML algorithms as opposed to feature engineering, where features are manually designed by humans. The hand-crafted features are subjected to human bias and in need of expert knowledge.

Deep learning is an advanced branch of the ML field that works upon the representation learning. It aims to discover the complex representation out of simpler representations. Deep learning methods are typically based on artificial neural networks that consist of multiple hidden layers with nonlinear processing units. The word 'deep' refers to the multiple hidden layers that are used for transforming the data representation. Using the concept of feature learning, each hidden layer of neural networks maps its input data into a new representation. The succeeding layer tends to capture a higher level of abstraction from the less abstract concept in the preceding layer. Therefore, a series of hidden layers increasingly extract more efficient features from the observed data. These high-level, abstract features obtained through successive layers are meant to be more informative for the ML task (e.g., classification and regression)[33]. Note that, in deep learning architectures, the hierarchy of learned features in multiple levels are finally mapped to the

output of the ML task in a unified framework.

Analogous to ML techniques, deep learning architectures are divided into two broad categories: (1) Unsupervised learning approaches including Restricted Boltzmann Machines (RBM), autoencoders, word embeddings, and Generative Adversarial Networks (GAN), (2) Supervised learning approaches including deep neural networks, Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). A large volume of published works by the deep learning community has been developed upon either these individual networks or a combination of them. The primary and active research fields that have exploited deep learning approaches are computer vision, natural language processing (NLP), and speech recognition. Some of the spectacular real-world applications of deep learning in these fields include image captioning, machine translation, text summarization, object classification in images, adding sounds to silent movies, and text generation. This dissertation seeks to contribute to this growing area of research by exploring the potential power of deep learning techniques in transportation-related applications.

1.3 Deep learning in Transportation

Statistical and ML methods have been exploited in transportation fields such as traffic flow prediction and car-following models for several decades. However, to the best of my knowledge, deep learning has emerged in transportation domains since 2014. Huang et al. [39] and IU et al. [40] were the first research groups who exploited deep learning approaches for traffic flow prediction. They demonstrated that their deep learning approaches can achieve higher prediction accuracy compared to the existing state-of-the-art. Afterward, several studies have been published on applications of deep learning in various transportation fields, where their main goal was to achieve a more efficient model against traditional approaches. In other words, deep learning architectures are not used because they are the only possible solution for a given task. They are utilized in the transportation fields that have already been solved by other technology-driven or data-driven methods, yet with the objective of enhancing the model performance. It is worth noting that deep learning techniques have been applied to only a small number of transporta-

tion applications such as traffic state prediction, signal timing, traffic accident, and transport mode inference. I personally exclude the fields of detecting and tracking traffic objects (e.g., vehicles, lanes, pedestrians, and traffic signs and lights) from images and videos as well as vision-based detection techniques for advanced driver assistant systems and autonomous vehicles from the list of transportation-related fields. Such fields are more connected to computer vision and image processing areas rather than traffic and transportation.

Much of the current literature in the application of deep learning in transportation domains pays particular attention to improving the accuracy of traffic flow and speed prediction models. Times-series models (e.g., ARIMA), probabilistic graph models (e.g., Markov chain), and nonparametric models (e.g. support vector regression) have been the widely used techniques for the problem of traffic flow and speed predictions [39]. Due to complex nature of the traffic flow process, researchers have been motivated to use deep learning algorithms that can represent traffic features in high levels of abstractions without prior knowledge. The examples of deployed architectures for learning traffic features include stacked autoencoder [53], deep belief network [39], combination of a linear model with deep neural network [62], long short-term memory (LSTM) network and a combination of CNN and LSTM [90].

Unlike traffic flow and speed prediction, there is a relatively small body of literature on deep learning for other domains of transportation including travel mode detection, vehicle classification, and traffic information system. Examining the efficacy of deep learning frameworks on such applications forms the core part of this dissertation.

1.4 Research contributions

In this dissertation, deep learning architectures are exploited for mining spatiotemporal information and crowdsourced information (e.g., GPS trajectories and Twitter texts) so as to address three transportation-related applications including travel mode detection, vehicle classification, and traffic information systems. The output of the research associated with this dissertation have been 4 papers published/-submitted into top-tier venues in three areas: 1) Transportation: Transportation

Research Part C: Emerging Technology, 2) Data mining: IEEE Transactions on Knowledge and Data Engineering, and 3) Artificial Intelligence: Expert Systems with Applications. In this section, I briefly highlight the scope and the major contributions of each paper. The details regarding each paper will be provided in the next chapters.

1.4.1 A CNN architecture for travel mode detection from GPS data

Identifying the distribution of users' transportation modes is an essential part of travel demand analysis and transportation planning. In this research (described in Chapter 2), a CNN-based model is proposed to predict the transportation mode(s) used in an individual's trip from their raw GPS trajectories, in which modes are categorized into walk, bike, bus, driving, and train. A user's raw GPS trajectory is defined as a sequence of time-stamped GPS points, each of which contains the information on the geographic location at a specific time. However, such an ordered list contains only a series of chronologically coordinate points without any explicit features and meaningful information. Furthermore, the structure of the raw GPS trajectory is not adaptable for deep learning algorithms. Accordingly, for the first time in this domain, a novel representation is designed for GPS trajectories in such a way that not only is adaptable with the deep learning algorithms but represents fundamental motion characteristics of a moving object including speed, acceleration, jerk, and bearing rate. This novel representation is then passed into a Convolutional Neural Network (CNN) architecture to predict traveler's transportation mode. The main contributions of this study are as follows:

- **Designing an efficient representation for raw GPS trajectories.** For the first time, a new procedure is developed for converting a raw GPS trajectory to an efficient and appropriate representation for using in deep learning architectures. The proposed representation contains the information of *all* GPS points in the GPS trajectory and allows the very deep architecture and training algorithm to extract discriminating and high-level features for the task-at-hand.
- **Developing a deep CNN architecture for identifying transporta-**

tion modes. An end-to-end deep learning architecture is developed to predict a user's transportation mode based on their raw GPS trajectory, which has been recorded while traveling around a traffic network.

- **Evaluating the proposed mode detection by comparison with several baselines.** Several standard machine learning techniques including ensemble algorithms and regular neural networks are developed for evaluation. Results clearly reveal that my proposed GPS representation and deep learning model outperform state-of-the-art transport mode detection models.

1.4.2 A deep learning architecture for vehicle classification from GPS data

Transportation agencies are starting to leverage increasingly-available GPS trajectory data to support their analyses and decision making. While this type of mobility data adds significant value to various analyses, one challenge that persists is lack of information about the type of vehicles that performed the recorded trips, which clearly limits the value of trajectory data in transportation system analysis. In this research (described in Chapter 3), a deep learning model is proposed to identify the vehicle's class from its trajectory, in which the vehicle class can be categorized as: light-duty (e.g., passenger cars), medium-duty (e.g., pick-ups), and heavy-duty (e.g., trucks). In comparison to my previous research, the GPS representation is significantly improved by involving more motion features as well as roadway features. Furthermore, an open source routing engine is deployed to obtain more accurate information on travel time and distance between GPS coordinates, which in turn results in a more efficient GPS representation. Before delving into training the proposed model, an efficient and scalable programmatic strategy is also designed to label 20 million GPS trajectories by means of vehicle information obtained through Virtual Weigh Station records. This paper makes the following contributions:

- **Labeling a large-scale GPS trajectory dataset based on the FHWA classification scheme**. An efficient programmatic approach is developed to label GPS trajectories by means of vehicle class information obtained from VWS vehicle records. Information from an open source routing

engine is also exploited to improve the accuracy of our labeling approach.

- **Designing a new representation for raw GPS trajectories**. First, a GPS trajectory is considered as a sequence of GPS legs, where each leg is the route segment between two consecutive GPS points. Then, a feature vector is computed for every GPS leg, which is a combination of motion-related and road-related features. Concatenating the feature vector corresponding to all GPS legs generates an efficient representation for each GPS trajectory.

- **Developing a deep Convolutional Neural Network for Vehicle-Classification (CNN-VC)**. For the first time in this domain, a CNN-based deep-learning model is developed to identify vehicle's class from the proposed GPS representation. The CNN-VC comprises a stack of CNN layers for extracting abstract features from the GPS representation, a pooling operation for encapsulating the most important information, and a softmax layer for performing the classification task.

- **Conducting an extensive set of experiments for evaluating the proposed model**. In addition to the classical machine-learning techniques, several state-of-the-art deep-learning algorithms are developed to be used as baselines. Experimental results reveal that our CNN-VC model clearly outperforms both classical-supervised algorithms and deep-learning architectures.

1.4.3 A semi-supervised deep learning approach for travel mode detection from GPS data

Almost all the current research work on travel mode identification and vehicle classification has built their models using only labeled trajectories, regardless of using shallow or deep learning algorithms. Nonetheless, a significant portion of GPS trajectories might not be annotated by a transport mode since the acquisition of labeled data is a more expensive and labor-intensive task in comparison with collecting unlabeled data. Thus, as an extension the first research, this study (described in Chapter 4) is proposing a semi-supervised deep convolutional network to harness both unlabeled and labeled GPS trajectories for predicting users'

transportation mode. The proposed semi-supervised model, which is more *algorithmic* compared to models in the previous research, integrates a convolutional-deconvolutional autoencoder and a convolutional neural network into a unified framework to concurrently perform supervised and unsupervised learning. The two components are simultaneously trained by minimizing a hybrid loss function that combines the losses of both supervised and unsupervised components. Furthermore, in the first research, it was assumed that users commute with only one transportation mode and in turn their associated GPS trajectory carries only one transportation mode. This assumption is relaxed by developing a novel trip segmentation process that partition a user's trip into GPS segments when the transportation mode changes. This paper makes the following contributions:

- **Developing a two-step segmentation process.** Due to the inherent need of CNN-based models for having a fixed-size input, the GPS trajectory of a trip is first uniformly partitioned into the GPS segments with a fixed size. Afterward, for the first time in this domain, a discrete optimization algorithm is deployed to detect the points where the transportation mode changes. The output of this step is a pool of GPS segments with only one transportation mode.

- **Developing a novel deep semi-supervised convolutional autoencoder architecture.** A deep semi-supervised model is proposed to leverage both *unlabeled* and labeled GPS trajectories for predicting transportation modes. The model contains Conv-AE and CNN classifier for unsupervised and supervised learning, respectively.

- **Building an effective schedule for tuning balancing parameters.** Since the main objective is to simultaneously training the unsupervised and supervised components of the proposed model, a novel and efficient schedule is proposed for varying the balancing parameters that combine reconstruction and classification losses.

- **Conducting an extensive set of experiments for performance evaluation and comparison.** The results reveal that my proposed model

outperforms several supervised and semi-supervised state-of-the-art baseline methods for various amounts of labeled GPS segments. Furthermore, the performance results demonstrate the superiority of the proposed trip segmentation process, the designed representation for GPS segments, the schedule for varying hyperparameters, and the model structure by comparing with several alternatives.

1.4.4 A traffic information system using Twitter data

As an application of natural language processing in ITS, a deep-learning architecture is developed for extracting real-time traffic congestion and incident events from Twitter streaming, which is described in Chapter 5. Since a tweet comprises a sequence of words and symbols with a variable length, they cannot be simply fed into ML methods as most of them require numerical features with a fixed size. Researchers have utilized the bag-of-words representation for converting tweets into numerical feature vectors. However, the bag-of-words not only ignores the order of tweet's words but suffers from the curse of dimensionality and sparsity. A common approach in literature for dimensionality reduction is to build the bag-of-words on the top of pre-defined traffic keywords. The immediate criticisms to such a strategy are that the pre-defined set of keywords may not include all traffic keywords and the tweet language is subjected to change over time. To address these shortcomings, word embeddings models are used to map tweets into a low-dimensional feature vector without any need of pre-defined traffic keywords. Word embeddings is an unsupervised learning algorithm that deploys a Neural Network Language Model (NNLM) to learn high-quality distributed representations of words in vector space by taking the semantic relationship between tweet words into account. Afterward, supervised deep-learning algorithms including convolutional neural network (CNN) and recurrent neural network (RNN) are deployed for detecting traffic-related tweets. This paper makes the following contributions:

- **Collecting and labeling a large volume of tweets.** More than 50,000 tweets are collected through Twitter APIs and labeled into three classes. Some shortcuts to increase efficiency are implemented in the labeling approach in order to increase speed without sacrificing the accuracy and quality of the

labeled data. To the best of our knowledge, the collected dataset is by far the largest labeled-tweet dataset in the traffic domain.

- **Developing a traffic event detection model using deep-learning architectures**. For the first time in this domain (i.e., traffic event detection from Twitter data), tweets are modeled into numerical feature vectors using word-embedding tools. Next, CNN and RNN are applied to tweet word-embedding matrixes for discriminating traffic-related tweets from non-traffic tweets.
- **Conducting an extensive set of experiments for performance evaluation and comparison.** First, proposed models in relevant studies are reproduced and used as the baselines. Comparison results clearly reveal that our proposed model outperforms the state-of-the-art models for detecting traffic-related tweets from Twitter data. Furthermore, we demonstrate the superiority of model settings and its capability for real-world situations through several other experiments.

The above-mentioned studies are elaborated in the next four chapters. The dissertation is concluded in Chapter 6.

Chapter 2

Inferring Transportation Modes from GPS Trajectories Using a Convolutional Neural Network

Abstract

Identifying the distribution of users' transportation modes is an essential part of travel demand analysis and transportation planning. With the advent of ubiquitous GPS-enabled devices (e.g., a smartphone), a cost-effective approach for inferring commuters' mobility mode(s) is to leverage their GPS trajectories. A majority of studies have proposed mode inference models based on hand-crafted features and traditional machine learning algorithms. However, manual features engender some major drawbacks including vulnerability to traffic and environmental conditions as well as possessing human's bias in creating efficient features. One way to overcome these issues is by utilizing Convolutional Neural Network (CNN) schemes that are capable of automatically driving high-level features from the raw input. Accordingly, in this paper, we take advantage of CNN architectures so as to predict travel modes based on only raw GPS trajectories, where the modes are labeled as walk, bike, bus, driving, and train. Our key contribution is designing the layout of the CNN's input layer in such a way that not only is adaptable with the CNN schemes but represents fundamental motion characteristics of a moving object including speed, acceleration, jerk, and bearing rate. Furthermore, we ameliorate the quality of GPS logs through several data preprocessing steps. Using the clean input layer, a variety of CNN configurations are evaluated to achieve the best CNN architecture. The highest accuracy of 84.8% has been achieved through the ensemble of the best CNN configuration. In this research, we contrast our methodology with traditional machine learning algorithms as well as the seminal

and most related studies to demonstrate the superiority of our framework.

Keywords: Convolutional neural network; Deep learning; GPS data; Transportation mode Inference

2.1 Introduction

Travel mode choice is one of the principal traveler's behavior attributes in travel demand analysis, transport planning, and traffic management. By inferring the travel mode distribution, transportation agencies are able to generate appropriate strategies to alleviate users' travel time, traffic congestion, and air pollution. For example, a clear benefit of travel mode analysis is to identify regions with high auto dependency and encourage public transport ridership by improving transit systems [23]. Furthermore, suitable policies such as HOV lanes can be taken during the peak-period congestion according to existing mode shares. The knowledge of the transport mode choice has traditionally been obtained through household surveys or phone interviews. However, interviewing households is a time-consuming and expensive method that usually results in a low response rate and incomplete information, which calls for a cost-effective technology such as Global Positioning System (GPS) that is able to collect travel data while reducing labor and time costs.

GPS is a ubiquitous positioning tool that records spatiotemporal information of moving objects carrying a GPS-enabled device (e.g., a smartphone). The main advantageous of smart phones, compared to other GPS-equipped devices, is its enormous market penetration rate in a large number of countries and being relatively close to users nearly all of the time. As a consequence, such a dominant and area-wide sensing technology is capable of creating massive trajectory data of vehicles and people. A GPS trajectory, also called movement, of an object is constructed by connecting GPS points of their GPS-enabled device. A GPS point, here, is denoted as (lat, long, t), where lat, long, and t are latitude, longitude, and timestamp, respectively. The study of individuals' mobility patterns from GPS datasets has led to a variety of behavioral applications including learning significant locations, anomaly detection, location-based activity recognition, and identification of transport modes [52], in which the latter is the focus of this

study. Nonetheless, GPS devices can only record time and positional characteristics of travels without any explicit information on utilized transport modes. This necessitates employing data processing and mining algorithms to extract hidden knowledge about transport modes from raw GPS data.

Many of proposed inference models on detecting transport modes by means of only GPS sensors include two steps. In the first step, a pool of attributes (e.g., velocity, acceleration, heading change rate, and stop rate) [97, 98], are computed from GPS logs. In the second step, the extracted features are fed into a learning algorithm to estimate the transportation mode. Unlike many classification problems that a majority of features have already been computed and included in the dataset, raw GPS trajectories contain only a series of chronologically ordered points without any explicit features such as speed and acceleration. This fact has required researchers to manually identify and formulate a set of features before using a machine learning technique for the classification task. However, the hand-crafted features may not necessarily distinguish between various transportation modes since they are vulnerable to traffic and environmental conditions [98]. Considering a congested traffic condition, for instance, the maximum velocity of a car might be equal to the bicycle and walk modes. To address this issue, one solution is to extract more features from a GPS track (e.g., top five velocities rather than a single maximum velocity). However, if a lot of features are generated to cover more aspects of GPS trajectories, the challenge of applying an effective dimensionality reduction process needs to be met. Manual features are typically produced based on feature engineering, which is a concept upon biased engineering justification and commonsense knowledge of the real world for creating features and making patterns more visible for learning algorithms. Since the ultimate performance of machine learning algorithms is contingent on how much accurate the hand-crafted features are, the effectiveness proof of such features is not trivial. One way to address the above-mentioned issues is to exploit deep learning algorithms that are able to automatically and without any human interference extract multiple levels of data representations.

Indeed, the input layer (i.e., input features) in deep learning architectures is the raw object (e.g., images) rather than a set of hand-crafted features. The key role of deep learning techniques is to encode object's raw and low-level features (e.g., raw

image pixels) to multiple levels of efficient and high-level features. This is the most salient attribute of deep learning algorithms that distinguishes them from classical machine learning algorithms. Thus, new representations of the raw data, which are more effective for the classification task, are generated by machines instead of humans. It should be noted that the learned features in the last layer play the same role as hand-crafted features, which are fed into activation functions (e.g., SVM or softmax) to compute class scores.

In this paper, we propose a Convolutional Neural Network (CNN) architecture to predict the transportation mode(s) used in an individual's trip from their raw GPS trajectories, in which modes are categorized into walk, bike, bus, driving, and train. The CNN is a type of deep learning techniques that has achieved great success in the fields of computer vision [48] and natural language processing [45]. We envision to investigate the capability of CNNs in representation learning and transport mode classification of raw GPS data. Unlike other fields (e.g., image classification) that images are easily utilized as the input layer, the key challenge in this study is to structure raw GPS tracks into a format that is not only acceptable for CNN architectures but efficient enough to represent fundamental motion characteristics of a moving object. In our methodology, an instance comprises four channels of kinematic features including speed, acceleration, jerk, and bearing rate. Stacking these channels yields a standard arrangement for the CNN scheme that also describes people's motion characteristics. After pre-processing data and designing a suitable layout for each instance, we come up with an effective CNN architecture so as to attain state-of-the-art accuracy on the GPS dataset collected by the Microsoft GeoLife project [98].

The rest of this article is organized as follows. After reviewing related works in Section 2, we set out details of our framework in Section 3, including preparing the input layer, applying data pre-processing steps, explaining settings of CNN layers, and generating several CNN configurations for our application. In Section 4, we evaluate our proposed CNN architecture on the GeoLife trajectory dataset. Comparing our results with classical machine learning algorithms and previous research is also carried out in Section 4. Finally, we conclude the paper in Section 5.

2.2 Literature review

A large and growing body of literature has proposed numerous frameworks for inferring the commuters' transport mode based on various data sources including raw GPS trajectories [24, 91, 97], mobile phone's accelerometers [58], and mobile phone's GSM data [77]. For performing the mode classification task, a wide range of traditional supervised mining algorithms have been applied, including rule-based methods, fuzzy logic, decision tree, Bayesian belief network, multilayer perceptron, and support vector machine [89]. Furthermore, some of the mobility detection research has integrated multiple sources to ameliorate the classification quality. For example, Stenneth et al. [79] exploited the GPS and GIS information for building their mode detection scheme while Feng and Timmermans [27] combined GPS and accelerometer data to generate a model that outperforms GPS-only information. In addition to the GPS and accelerometer, the data from other mobile phone sensors such as gyroscope, rotation vector, and magnetometer have been deployed in distinguishing between different transportation modes [22, 41]. Integrating such spatial and temporal information with socio-demographic characteristics of travelers can also lead to generating richer travel mode detection models [6]. However, the methodologies using only one type of sensor can be more practical since accessing to several data sources may not be possible in many cities [91]. As this study seeks to design a mobility inference model using only raw GPS data, we only review the studies that examine the capability of the GPS sensor for distinguishing between users' transportation modes. A comprehensive and systematic review of existing techniques of travel mode recognition based on GPS data is available in the reference [89]. The paper provides an excellent comparison of various approaches in three categories including GPS data preprocessing, trip/segmentation identification, and travel mode detection.

Zheng et al. [98] proposed a solid framework to automatically infer transportation mode(s) from users' GPS trajectories. In their seminal study, first, a change-point-based segmentation algorithm is applied to divide a trip into segments with distinct transportation modes. Afterward, features of each segment are extracted, including the mean and variance of the velocity, the expectation of velocity, the top three velocities and accelerations. Considering each segment with its features and

its assigned transportation label as an independent instance, various traditional classification algorithms (e.g., decision tree) were deployed to train the mode inference model. In a follow up study, they improved the performance accuracy of their mode inference framework by involving more robust trajectory's attributes as well as applying a graph-based post-processing algorithm [97]. The new features are composed of the rate of change in the heading direction, the stop rate, and the velocity change rate. In the postprocessing algorithm, a graph is built based on clustering users' change points into nodes and assigning the probability distribution of different modes on edges. Using the GPS data, Sun and Ban [80] utilized the features related to only accelerations and decelerations (e.g., the proportions of accelerations and decelerations larger than 1 meter per square second, and the standard deviations of accelerations and decelerations) to classify vehicles into general trucks and passenger cars. A recent study by Xiao et al. [91] demonstrated that tree-based ensemble classification algorithms outperform traditional ones such as the decision tree. Their major contribution was to augment the number of GPS trajectory features to 111 using statistical methods. The features were categorized into global attributes, extracted from descriptive statistics for the entire trajectory, and local features, extracted from the profile decomposition that describes the movement behavior. Mäenpää et al. [57] sought to select the most significant features from three sets of potential features extracted from GPS trajectories. According to the statistical tests, frequency-domain features were found to be significant in classifying transportation modes while auto- and cross-correlations, kurtoses and skewnesses of speed and acceleration were not useful.

A few recent studies have attempted to integrate the hand-crafted features developed by Zheng et al. [97] with high-level features extracted through deep learning architectures [24, 85]. Endo et al. [24] exploited a fully connected Deep Neural Network (DNN) to automatically extract high-level features. Their core idea was to convert a raw GPS trajectory into a 2-D image structure as the input for the DNN model. The pixel value in the created trajectory image is equivalent to the duration time that a user stays in the location of the pixel. After integrating the image-based deep features with the manual features, a traditional classifier is deployed to predict transportation labels. However, the method of creating trajectory images suffers from some drawbacks. The pixel values contain only the duration time without any motion attributes and a theoretical concept. In other

words, the technique fails to take spatiotemporal information into account that can be measured by manipulation of latitude, longitude, and timestamp. Lack of any motion information (e.g., speed and acceleration) in the input layer leads to generating irrelative features to the mission of mode inference. Moreover, in order to prevent having biased pixel values, only GPS points with a fixed time interval between two consecutive GPS points are sampled, which results in losing valuable information.

In a similar study conducted by Wang et al. [85], deep features are obtained by transforming point-level features (PF) with the aid of the sparse auto-encoder. They defined PF as a time series of speed, head change, time interval, and distance of the GPS points. The deep features are then aggregated using a simple CNN before being combined with the hand-crafted features. Finally, both types of features are fed into a DNN to infer the transportation mode. Although their work involved the motion characteristics in the original input features, multicollinearity arises from a high and linear correlation between speed, time interval, and distance. This issue degrades the quality of the feature learning in auto-encoder and CNN techniques. Furthermore, the distance and time interval obtained by means of GPS-equipped devices are not discernable features for detecting transport modes. For instance, a car moving in a crowded area with huge skyscrapers may record two consecutive GPS points with a large distance while a walking individual can record a short distance in the environment with no signal blockages. Note that the distance of two successive GPS points for the car is shorter than the walk mode in a normal situation. It is worth noticing that the highest accuracy of both deep-learning-based studies [24, 85] are still lower than the work of [98], which is purely based on manual features.

To tackle the above-mentioned shortcomings, we propose an efficient CNN architecture with various types of layers that receive an informative input layer with kinematic characteristics. Both feature learning and classification task are executed in an integrated CNN architecture. To show the superiority of CNNs, we also compare the predication quality of our CNN scheme with traditional supervised learning algorithms that have broadly been used in transportation-mode-inference models.

2.3 Methodology

2.3.1 Preparing the input samples and applying data processing

The raw GPS data for each user consists of chronologically ordered points that have been collected over a time period. First, the user's GPS track is divided into trips if the time interval between two consecutive GPS points exceeds a pre-defined threshold. Each trip is divided into segments based on the change in the transportation mode (i.e., the GPS series of each segment contains only one transportation label). In the CNN architectures, all samples are required to have the same size. Considering a constant length for all instances, each segment is either sub-divided into a pre-defined number of GPS points or padded with zero values to have the same size as other segments. After generating segments with the same size, we compute the motion characteristics of each GPS point using the tuple (lat, long, t) of two consecutive points.

We utilize the Vincenty's formula [84] for calculating the geographical distance between two succeeding GPS points P_1 and P_2. Denoting the time difference between P_1 and P_2 as Δt, the first three motion features of P_1 are then calculated based on the following equations:

$$S_{p_1} = \frac{RD_{p_1}}{\Delta t_{p_1}} \tag{2.1}$$

$$A_{p_1} = \frac{S_{p_2} - S_{p_1}}{\Delta t_{p_1}} \tag{2.2}$$

$$J_{p_1} = \frac{A_{p_2} - A_{p_1}}{\Delta t_{p_1}} \tag{2.3}$$

where S_p, A_p, and J_p represent the speed, acceleration/deceleration, and jerk of the point P, 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 [4]. Accordingly, we engage it in our calculation as a distinguishable attribute among various modes.

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 [97]. To quantify this disparity among modes, we introduce the bearing rate as the fourth motion attribute. 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:

$$y = \sin\big(p_2[lon] - p_1[lon]\big) \times \cos\big(p_2[lat]\big) \tag{2.4}$$

$$x = \cos([p_1[lat]) \times \sin([p_2[lat]) - \sin(p_1[lat]) \tag{2.5}$$

$$Bearing_{p_1} = \arctan(y, x) \tag{2.6}$$

$$BR_{p_1} = \mid Bearing_{p_2} - Bearing_{p_1} \mid \tag{2.7}$$

where sine, cosine, and arctangent are trigonometric functions. Latitude (lat) and longitude (long) needs to be in radians before passing to Eqs. (4)–(6). The output of Eq. (6) is then converted to degree. According to Eqs. (4)–(7) and Fig. 4.2, the bearing rate (BR) is calculated using the information of three consecutive GPS points. Analogously, the above-mentioned formulas are utilized to calculate the motion features of other GPS points in a segment.

Before designing an adaptable input layer for CNN architectures, we need to identify and remove outliers and random errors that have been generated due to several error sources such as satellite or receiver clocks, atmospheric disturbances, and multipath signal reflection. First, we apply the following data preprocessing steps to detect and remove invalid and inaccurate GPS points:

- The GPS data points with the timestamp greater than their next GPS point are identified and discarded.

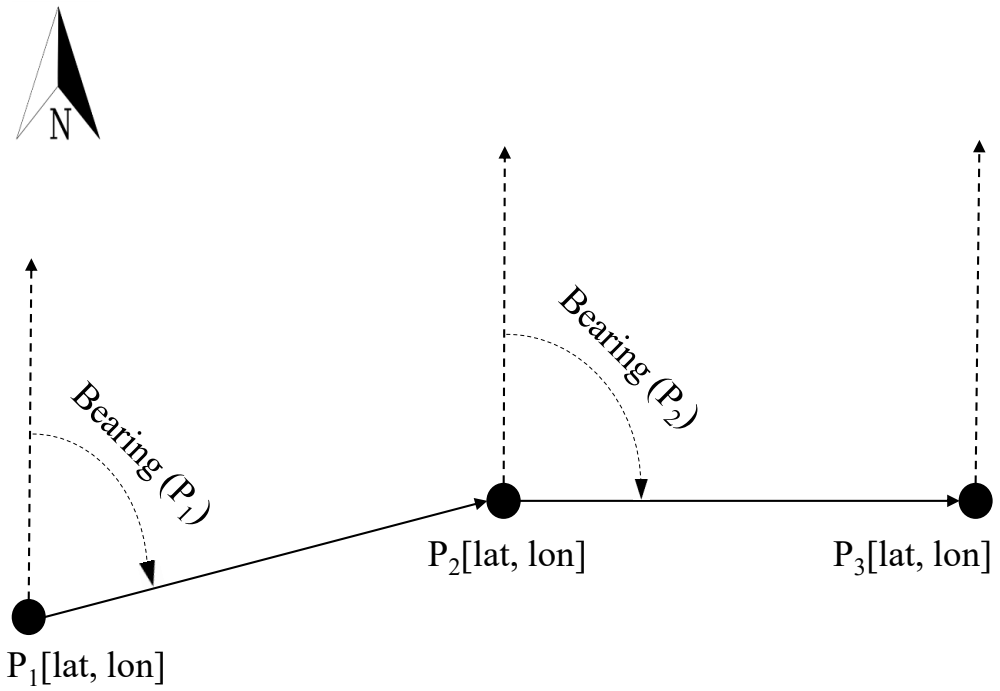


Figure 2.1: Bearing between two consecutive GPS points.

- Considering a maximum threshold speed for each transport mode, provided in Table 2, the invalid GPS points with an unrealistic large speed are discarded.
- Considering a maximum threshold acceleration for each transport mode, provided in Table 2, the invalid GPS points with an unrealistic large acceleration are discarded.
- After removing the unrealistic GPS points, the segments with the number of GPS points less than a threshold are identified and discarded.

In the second step of data pre-processing, we apply a smoothing kernel to GPS trajectories so as to remove random errors. Smoothing is a process that data points are averaged by their neighbors using a specific kernel function, in which the shape of kernel distinguishes various smoothing techniques. Since we have no sense on how the sequence of motion characteristics for a segment might fluctuate, we need to utilize a kernel that has no pre-defined shape. In this paper, we implement the Savitzky-Golay filter to all segments' sequences. For each data point of a sequence, the method fits a polynomial to a set of input samples laid over an

odd-sized window centered at the subject point [70]. Then, the new value of the subject point is computed by evaluating the resulting polynomial at the subject point. The main advantageous of Savitzky-Golay filter is its capability to maintain the original shape and pattern of the signal.

After obtaining the clean and validated segments with high-quality GPS points, we must stack the vectors of the motion features related to each segment so as to create independent samples. Accordingly, a four-channel structure is built for each segment, where each channel represents speed, acceleration/deceleration, jerk, and bearing rate. Thus far, we have achieved to prepare input samples for feeding into CNN architectures. In the next section, we describe our designated CNN configurations that are primarily inspired by Simonyan and Zisserman [75] and Krizhevsky et al. [48].

2.3.2 CNN architecture

Although the key idea in the Convolutional Neural Network (CNN) is similar to the ordinary feed-forward artificial neural network, they differ in terms of connectivity patterns between the neurons in adjacent layers. Unlike the traditional multilayer perceptron (MLP) that each node is fully connected to nodes in the previous layer, the CNN takes the advantage of the spatially local correlation by connecting neurons to only a small region of the preceding layer, also called the receptive field. Such a local connectivity between nodes leads to having the smaller number of weights, which mitigates the curse of dimensionality and the overfitting problem.

Typically, a CNN architecture constitutes a sequence of layers in which each layer transforms an input volume to an output volume of neurons using a set of operations. Although various types of layers have been introduced in literature, we describe only those layers that have been used in building our CNN structure, including the input layer, convolutional layer, pooling layer, fullyconnected layer, and dropout layer.

2.3.2.1 Input layer One distinguishing feature in the CNN is the capability of accepting the input volume in three dimensions: height, width, and depth (channels). The input layer comprises a set of independent samples, where each sample

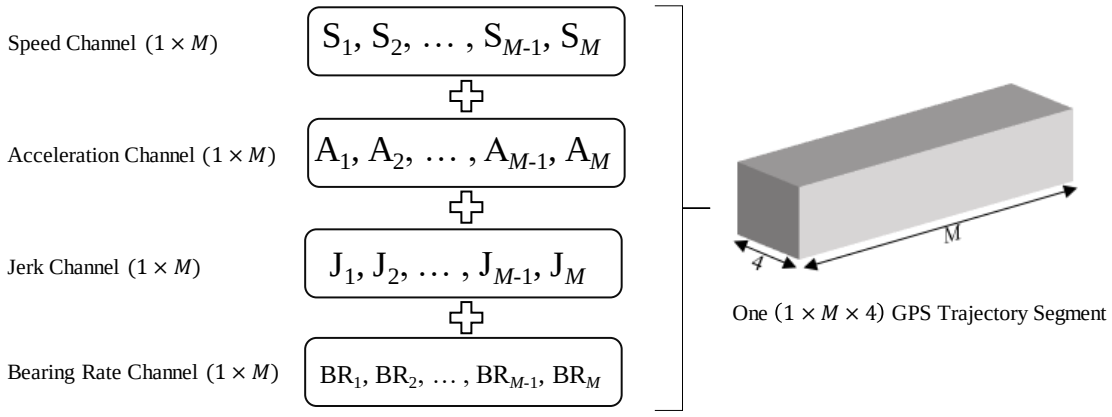


Figure 2.2: The four-channel structure for a GPS segment (i.e., an independent sample).

is a GPS segment with four channels including speed, acceleration, jerk, and bearing rate. Each channel has the shape of $(1 \times M)$, where M determines the segment length (i.e., the number of GPS points that forms the segment). Thus, the input shape is a tuple of $(1 \times M \times 4)$. Since the GPS segments contain a different number of GPS points, the value of M may vary over the samples. On the other hand, in the CNN, the input shape for all instances must be the same. For addressing this challenge, all segments are restricted to a fixed size of M. Thus, long segments are truncated to M while the shorter ones are padded with zero values. Fig. 2.2 illustrates the four-channel structure for a GPS segment.

2.3.2.2 Convolutional layer The convolutional layer comprises a set of learnable filters. Each neuron in the convolutional layer's output is connected to the small region (receptive field) of the previous layer, where the size of the receptive field is equivalent to the filter. The output value of the neuron is computed by operating dot product between the parameters of the filter and the entries of its receptive field. Convolving the same filter across the whole surface of the input volume creates a 2-dimensional map in the output volume, also called feature map or activation map. Performing similar operations for all layer's filters creates several activation maps. Stacking the feature maps of all filters along the depth dimension creates the 3-d output volume of the layer. Since the size of the input volume is small in our application, we use a small $(1 \times 3 \times C)$ convolution filters throughout for all convolutional layers. For each layer, C indicates the number of channels in the layer's input volume. Using smaller receptive fields leads to reducing the

number of parameters and mitigating the overfitting problem [75].

The 3-d output shape of each convolutional layer is controlled by using three hyperparameters, which needs to be specified by the user. The depth of the output volume, denoted as D, is the first parameter that corresponds to the number of filters used for the convolutional operation. Stride, denoted as S, is the number of elements by which the filter is moved at a time over the input volume. The zero-padding, denoted as P, is the processing of adding zero values at the start and end of the input volume matrix so as to control the spatial size of the output. In all convolutional layers, we set $S = 1$ and set P so that the spatial dimensions of the layer's input do not alter (i.e., the layer's input and output volumes have the same size). But, the number of filters are tuned through a manual search over a variety of CNN configurations as described in previous sections.

2.3.2.3 Activation layer After obtaining the convolved output volume, the neurons are often followed by an activation operation with the purpose of introducing the nonlinearity in the model. Among several types of activation function, we utilize the non-saturating nonlinearity $f(x) = max(0, x)$ in all convolutional layers, where x denotes the convolved neurons. The function, which is referred to the Rectified Linear Units (ReLU), replaces all negative values in the feature map by zero. Compared to other functions such as $tanh(x)$ and $sigmoid$ function, the learning rate of the CNN with ReLU is much faster [48].

2.3.2.4 Pooling layer The objective of pooling layer is to achieve spatial and scale invariance, lower computation, and controlling overfitting by decreasing the dimensionality of each feature map and down sampling the convolution layer spatially [71]. Max-pooling is the most common kind of spatial pooling that partitions each depth slice of the input volume into non-overlapping vectors, then take the maximum value in each sub-vector. The filter size of the max-pooling layer determines the length of sub-vectors (i.e., the number of entries that the max should be taken over). Hence, the depth of input and output volumes in the pooling layer are the same. In our problem, we insert the max-pooling layer with the filter size (1×2) and $S = 1$ periodically between the successive convolutional layers.

2.3.2.5 Fully-connected layer The CNN architecture can be brought to an end with several Fully-Connected (FC) layers. Like the traditional multilayer percep-

tron, every neuron in the FC layer is connected to all neurons in the previous layer and computed by the element-wise multiplication. Except for the last FC layer, the rest play the role of extracting features. Subsequently, the extracted high-level features from previous layers are fed into the last FC layer for performing the classification task, in which the $softmax$ activation function is utilized to generate a probability distribution over the transportation labels. Except for the last FC layer, it is not necessary to have the number of the output neurons in FC layers the same as the number of labels.

2.3.2.6 Regularization Regularization is a process used in an attempt to prevent the overfitting problem in statistical models by explicitly controlling the model complexity and constraining the fitting procedure [28]. Overfitting is a major problem in the CNNs due to a large number of weights and complicated relationships between inputs and outputs. Dropout is the most practical and widely used approach to overcome the overfitting problem in CNNs [28]. The technique drops out the input units, with all its incoming and outgoing connections, with a probability of P from the network at each update during the training process. Only the parameters associated to the reduced network is trained at each stage.

Increasing the number of samples is another approach to prevent the overfitting problem. The idea of subdividing the segments into samples with the fixed length M is also augmenting the samples more than four times compared to the previous studies [24, 91, 97]. Since the CNN typically work well in large-scale datasets, the data augmentation is an effective strategy due to lack of a massive GPS data set with transportation labels.

Furthermore, we implement the idea behind the bootstrap aggregating algorithm that combines the output of multiple base learners [9]. In our application, the base learner is the best CNN configurations. Thus, the base learners are first trained independently on randomly selected training instances with replacement, and then their $softmax$ class probabilities are averaged to generate the transportation label posteriors.

2.3.2.7 CNN configurations Setting the size of filters and stride the same as what described above for all layers, a variety of CNN configurations can be built based on the number of layers, the order of layers, and the number of filters in each

convolutional layer. We design a broad spectrum of networks to achieve the best one that fits well to our application. Yet, instead of applying an exhaustive and computationally expensive search over all CNN's hyperparameters for identifying their optimal values, we pursue an efficient manual search inspired by one of the most cited papers in the CNN field [75]. We start with shallow networks and low number of filters and gradually increase the number of layers and filters in each layer to evaluate if the CNN model's beats higher accuracy. For the sake of simplicity, as illustrated in Table 2.1, we bring only the important configurations out of many tested networks to highlight the effect of four hyperparameters, including the layers' pattern, the absence or presence of a layer, the CNN's depth, and the layers' number of filters, on the prediction quality. Initially, we set the P values in dropout layers as 0.5. Yet the final optimal values of P in dropout layers are optimized using the grid search method. The overall layout stacks two convolutional layers, following with or without max-pooling and dropout layers, and may repeat this pattern for a few times to increase the CNN's depth. The configurations are completed by adding one to three FC layers. Excluding the last FC layer, outputs of all convolutional and FC layers are activated by $ReLU$. Apart from the last FC layer, the number of neurons in an FC layer are equal to one-fourth of neurons in its previous flattened layer.

2.3.2.8 Training process The goal in the training process is to learn parameters of layers' filters in such a way that a loss function is minimized. We utilize the categorical cross-entropy as the loss function to compute the error in the output layer. We use the Adam optimizer to update model parameters in the back propagation process. Adam is a well-suited optimization technique for problems with large dataset and parameters that has recently seen broader adoption for deep learning applications [46]. We use the batch size equal to 64 and Adam's default settings as provided in their paper: $learningrate = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We initialize the parameters in the convolutional and fully connected layers by following the proposed scheme in [30]. In our initial analysis, we use a low number of epochs equal to 10 for identifying the optimal CNN configuration in Table 2.1 and the optimal combination of P values in dropout layers. Afterward, we apply the early stopping method to identify the optimal number of epochs for training the best identified CNN net, which avoids overfitting problem.

Table 2.1: Multiple CNN configurations per column. The number in the convolutional layers is the number of filters. Filter sizes for all convolutional layers and max-pooling layers are (1×3) and (1×2), respectively. Number of neurons in FC layers is one-fourth of neurons in its previous flattened layer. Number of neurons in the last FC layer is 5, equal to the number classes. Yes and No indicate the existence or non-existence of a layer. Fraction rate of dropped units in the FC layers is 0.5. Bold elements show the change(s) compared to the previous configuration.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Input layer	**A pool of** $(1 \times 200 \times 4)$ **GPS segments**								
Convolutional	32	32	32	32	32	32	32	32	32
Convolutional	32	32	32	32	32	32	32	32	32
Max-pooling	No	No	No	No	**Yes**	Yes	Yes	Yes	Yes
Dropout	No	No	No	No	No	**Yes**	**No**	No	No
Convolutional	No	**64**	64	64	64	64	64	64	64
Convolutional	No	**64**	64	64	64	64	64	64	64
Max-pooling	No	No	No	No	**Yes**	Yes	Yes	Yes	Yes
Dropout	No	No	No	No	No	**Yes**	**No**	No	No
Convolutional	No	No	**128**	128	128	128	128	128	128
Convolutional	No	No	**128**	128	128	128	128	128	128
Max-pooling	No	No	No	No	**Yes**	Yes	Yes	Yes	Yes
Dropout	No	No	No	No	No	**Yes**	Yes	Yes	Yes
Convolutional	No	No	No	No	No	No	No	**256**	**No**
Convolutional	No	No	No	No	No	No	No	**256**	**No**
Max-pooling	No	No	No	No	No	No	No	No	No
Dropout	No	No	No	No	No	No	No	No	No
FC	No	No	No	No	No	No	No	No	**Yes**
Dropout	No	No	No	No	No	No	No	No	**Yes**
FC	No	No	No	**Yes**	**Yes**	Yes	Yes	Yes	Yes
Dropout	No	No	No	No	No	**Yes**	Yes	Yes	Yes
FC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

In the early stopping method, the training and validation scores (e.g., accuracy) are computed after each epoch of training. The number of epochs that results in the maximum validation score is selected as the optimal value of epochs.

2.4 Data description and creation of GPS segments

The proposed methodology is examined and validated on the GPS trajectories collected by 69 users in the GeoLife project [99]. Although many kinds of transport modes have been labeled by users, we only consider the ground transportation modes. In accordance with the user guideline linked to the published dataset [99], we assign driving as the label of both taxi and car. Also, we refer all reported rail-based modes (e.g., subway, train, and railway) to train due to the layout of the rail-based system explained in the user guide. Therefore, our final list of transportations modes is: walk, bike, bus, driving, and train. After matching each user's label file with its corresponding GPS trajectories file, the user's GPS track is divided into trips if the time interval between two consecutive GPS points exceeds twenty minutes as the interval threshold [98]. Next, trips are converted to into fixed-size segments when the number of GPS points for each segment is set to $M = 200$. The number 200 is the median of the number of GPS points in all trips. This is also used as the segmentation procedure for dividing the unseen GPS tracks before the classification task. So our proposed method is capable of classifying a GPS track that lasts approximately 10 min since more than 91.5% of trajectories has been recorded in a dense representation (e.g., every 1–5 s) [99]. The two consecutive segments with an identical label are then concatenated together. After calculating the motion characteristics of each GPS point, all the data preprocessing steps, described in the previous section, are applied to the created segments. The maximum allowable speed and acceleration pertaining to each mode are provided in Table 2.2. Using several reliable online sources and the engineering justification (e.g., existing speed limits, current vehicle and human's power), the speed and acceleration thresholds are designated so as to reject only the GPS points with unlikely speed and acceleration values. The segments with less than 10 GPS points are discarded. The window size and polynomial order in the Savitzky-Golay filter are set to 9 and 3, respectively. The distribution of the created samples among the modes are also illustrated in Table 2.2, with the total number of segments equal to 32,444.

Table 2.2: Number of samples, maximum speed, and maximum acceleration associated with each transportation mode

Transportation Mode	**Number of Segments**	**Max. Speed** (m/s)	**Max. Acceleration** (m/s^2)
Walk	10372	7	3
Bike	5568	12	3
Bus	7292	34	2
Driving	4490	50	10
Train	4722	34	3

2.5 Result and discussion

All data processing has been coded in the Python programming language. The CNN architectures have been implemented in Keras [14], a Python-based deep learning library, using the TensorFlow backend with the CPU support only. We randomly sample 80% of the whole created segments as the training set while holding out the rest as the testing data. The final performance evaluation of a CNN model needs to be done only on the test set that plays no role in the training process.

2.5.1 Identifying the optimal CNN configuration

In the first round of our experiment, we seek for an optimal CNN configuration in terms of the layers' pattern, the absence or presence of a layer, the CNN's depth, and the layers' number of filters. The test accuracy rates of the most important CNN arrangements in Table 2.1 are set out in Table 2.3.

In the first three configurations, we only use convolutional layers. Increasing the number of convolutional layers from 2 in the model A to 6 in the model C enhance the accuracy of model by nearly 2%. Note that we increase the number of filters while the CNN's depth rises so as to capture more abstract features in last layers before the classification task. Adding an FC layer in the model D cannot improve the performance by itself. In order to assess the effect of max-pooling layers, we add a max-pooling layer right after each group of convolutional layers in the network E, which increase the test accuracy a bit. Analogously, we insert dropout layers after max-pooling and FC layers to avoid potential overfitting in the network F.

As can be seen in Table 2.3, the accuracy dramatically goes down to 69% from the previous best performance 77% in the model E. The rationale behind such a significant decrease in accuracy is that using many dropout layers simplifies model too much and in turn causes high bias. Accordingly, a proper arrangement of dropout layers can make a balance between bias and variance (i.e., overfitting and underfitting problems). So we remove the dropout layers in the first groups of convolutional layers, which results in increasing the accuracy to its highest value of 79.8% in the model G. Keep the layout of model G, we also create deeper and more complex CNNs in models H and I by adding more convolutional and FC layers. In spite of achieving nearly similar results, the test accuracy has not improved in comparison with the configuration G. Another advantage of the network G is its less computational cost due to being shallower than models H and I.

In the second round of our experiment, we attempt to ameliorate the best CNN configuration G by first optimizing the P values in the dropout layers and then identifying the proper number of epochs for training. We specify the range of P values for both dropout layers in the model G as $[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$. By exhaustively considering all combinations of P values and using a 5-fold cross validation over the training set, we obtain the highest score when both P values are set to 0.5, the same as the original model G. To obtain the optimal number of epochs, we train the model G over 120 epochs while the model performance is computed on both training and test sets at the end of each epoch. The optimal number of epochs is the point when no further increase occurs on the test score. The method is also known as early stopping. Accordingly, Fig. 2.3 compares the test and training scores for varying number of epochs. As expected, the training accuracy goes up by increasing the number of epochs and peaks at almost 96%. However, the test accuracy as the main evaluation metric levels off around 81–82% after around 30 epochs of training. The highest test accuracy is 82.3%, achieved with 62 epochs of training. Since the test accuracy remains almost constant and does not drop with increasing the number of epochs, we ensure the overfitting problem does not occur in our CNN model. The main remedy for a machine learning algorithm that performs well on training set but not on test set is to increase the training data. In other words, lack of training data in our application is the principal reason for inability to enhance the test accuracy while the model performs well on the training set yet with no overfitting problem. Also, the training

time is affordable. It takes about 25 minutes to train the model G for 62 epochs using a system equipped with a Core i7 2.50 GHz processor and a 16.0 GB memory. As a consequence, we select the single configuration G with its optimized P values, which has been trained by 62 epochs, as the excellent architecture for applying the ensemble concept. We train seven CNNs with the same pattern of the model G but on separate training sets that have randomly been sampled with replacement from the original training set. Our ensemble yields the highest accuracy of 84.8%, a 2.5% increase compared to the model G. The results of test accuracy for different models in the second round of our experiment are summarized in Table 2.4.

Table 2.3: Test accuracy rate of the CNN configurations in Table 2.1.

CNN Configuration	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Accuracy (%)	74.5	75.0	76.9	75.1	77.0	69.2	**79.8**	79.6	78.3

Table 2.4: Test accuracy of various versions of the best CNN configuration (model G)

Improvement steps for the best CNN net (model G)	**Test Accuracy (%)**
Original model G in 2.1	79.8
Model G with optimal P values of dropout layers	79.8
Model G trained with optimal number of epochs 62	82.3
Ensemble of model G	84.8

Table 2.5 provides the details of our best CNN performance (i.e., the ensemble of the model G) including the confusion matrix, recall, and precision pertaining to each transportation mode. Recall for each mode implies the accuracy of the predictor for only that mode. However, precision is the fraction of true instances among all predicted instances for each mode. In general, the results, particularly the recall rates, demonstrate the effectiveness of the CNN architecture in inferring the transportation modes, in which all precision and recall values exceed 67%. As reported in Table 5, there is a high correlation between the performance of the model in predicting a mode and the number of available instances for that mode. A large volume of samples and exclusive features of the walk mode results in achieving a perfect recall of 95.7%, whereas the driving mode with the lowest number of available segments obtains the lowest accuracy of 67.4%. It should be noted that the travelers' behavior in the driving mode is much more unpredictable since a

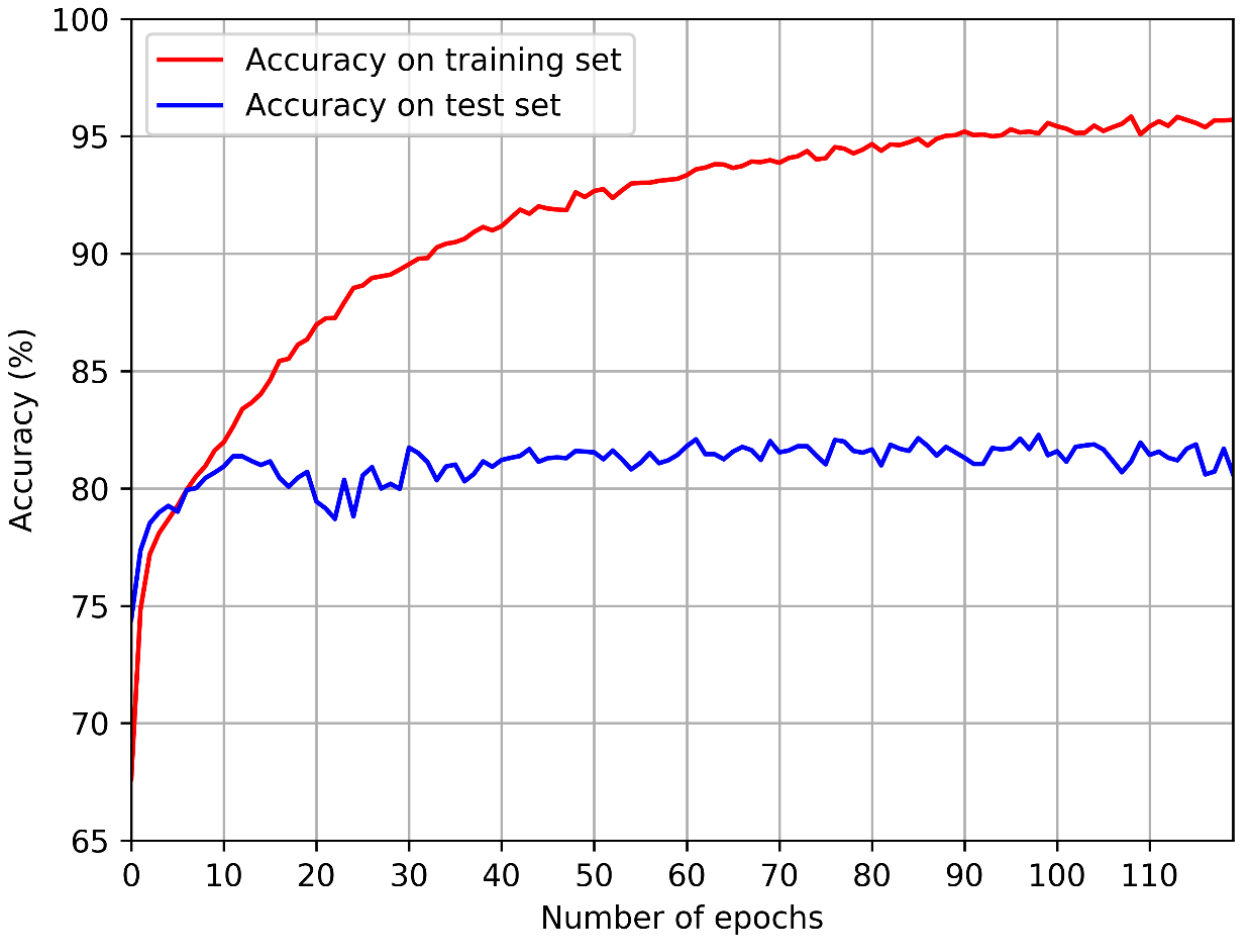


Figure 2.3: Comparing the test and training accuracy for varying number of epochs on the best CNN configuration (model G).

wider range of taxi and passenger cars exist that have more flexibility in maneuvers compared to other vehicles. The bus and train are the transit modes that must adhere to pre-defined routes and schedule, which leads to more predictable mobility behavior. Another interesting yet reasonable finding is that a large portion of false negative instances for the driving mode is bus, 171 out of 902. This highlights that the CNN topology requires more instances to distinguish between the driving mode and the bus, as the most identical mode to the car and taxi.

Table 2.5: Confusion matrix, recall, and precision for the ensemble of the model G.

Ensemble of the model G		**Predicted Class**						
		Walk	**Bike**	**Bus**	**Driving**	**Train**	**Sum**	**Recall%**
Actual Class	**Walk**	2014	53	40	3	2	2112	95.7
	Bike	171	946	32	3	1	1153	82.6
	Bus	147	23	1104	70	25	1369	81.1
	Driving	71	17	171	601	42	902	67.4
	Train	75	17	31	24	806	953	85.3
	Sum	2478	1056	1378	701	866	6489	-
	Precision%	81.6	90.3	80.7	86.6	92.3	-	-

2.5.2 Comparison with classical machine learning algorithms

In order to assess the performance of our best CNN model in comparison with classical machine learning algorithms, we choose widely used methods for learning transportation modes based on hand-crafted features, including K-Nearest Neighborhood (KNN), RBF-based Support Vector Machine (SVM), and Decision Tree (DT). We also compare our CNN with Random Forests (RF), as a representative of ensemble algorithms, and Multilayer Perceptron (MLP), as a regular and fully connected neural networks. All models are implemented and evaluated using scikit-learn, a Python-based machine learning library.

To have a fair comparison, the traditional inferring algorithms are trained and tested using the same training and test trajectory segments as were utilized in the CNN model. However, the input features for each segment needs to be manually designed. We utilize the same features as introduced in [97, 98], including the segment's length, mean speed, expectation of speed, variance of speed, top three speeds, top three accelerations, heading change rate, stop rate, and speed change rate. These features are the most acceptable GPS trajectories' attributes that have been utilized by many researchers.

Using the grid search method and 5-fold cross validation on the training set, the estimators' hyperparameters are optimized to ensure a good fit of the data and have the best trained model. The most important hyperparameters are enumerated as the number of neighbors for KNN, the maximum depth of tree for DT,

the regularization parameter for SVM, the number of weak learners (i.e., trees) for RF, and the number of hidden layers for MLP. In the MLP model, we set the number of neurons in the preceding hidden layer twice as the number of neurons in its previous layer. After tuning the hyperparameters through an exhaustive search, the optimal estimators are evaluated on the test set. Table 2.6 compares the performance quality of our best CNN model with the above listed supervised learning algorithms on the similar test data in terms of four classification metrics: test accuracy, average precision, average recall, and average F-score. It should be noted that the reported precision, recall, and F-score are the average among all modes of transportation. The optimal hyperparameter of each estimator over the specified parameter range is also shown in Table 2.6. The comparison proves the superiority of our CNN model, in which the test accuracy of our best CNN is almost 16% higher than the average test accuracy of other methods. Almost the same results have been obtained for other performance measures. We also seek to compare the prediction quality of the shallowest and simplest CNN configuration in Table 2.1 (i.e., CNN-A) with other classical techniques. With regard to Table 2.6, it is interesting that the tuned CNN-A with only two convolutional layers still outperforms KNN, SVM, DT, and MLP models. The test accuracy of the RF model, as the best classic model, is better than the shallowest CNN model by only 2.5%. Another interesting finding is that our best CNN and the shallowest CNN significantly outperform the regular neural network (i.e., MLP). As was previously mentioned, the power of the CNN algorithm in extracting higher-level features through multiple layers of nonlinear processing units results in outperforming the traditional supervised learning algorithms that are fitted based on feature engineering.

Table 2.6: Comparison of CNN's inference accuracy with five classical machine learning algorithms. HP: Hyperparameter, NA: Not applicable.

Model	**HP**	**HP Range**	**Optimal HP**	**Accuracy**	**Precision**	**Recall**	**F-score**
KNN	No. of neighbors	[3, 40]	5	63.5	63.2	63.5	62.4
SVM	Regularization	[0.5, 20]	4	65.4	66.9	65.4	64.9
DT	Tree Depth	[1, 40]	10	75.2	75.2	75.2	74.9
RT	No. of trees	[5, 100]	85	78.1	78.2	78.1	77.7
MLP	No. layers	[1, 10]	1	59.4	58.8	59.4	54.6
CNN-A	Satisfied	NA	NA	75.6	75.2	75.6	74.8
Best-CNN	Satisfied	NA	NA	**84.4**	**86.3**	**82.4**	**83.9**

2.5.3 Comparison with the previous studies

To show the superiority of our model, we compare our results with the studies that have used the GeoLife GPS trajectory dataset in order to establish a fair comparison. We choose the seminal study of Zheng et al. [98], who developed a solid inference mode framework for the first time in 2008, and the studies that employed the deep learning approaches to extract high-level features: Endo et al. [24] and Wang et al. [85]. Table 2.7 compares the prediction performance of our best CNN model with the highest performance accuracy reported by these studies. As can be seen in Table 2.7, our best CNN architecture significantly outperforms the frameworks in the mentioned studies by improving the accuracy more than 8%, 16%, and 10% compared to the work of Zheng et al., Endo et al., and Wang et al., respectively. This cutting-edge accuracy stems from our data pre-processing steps, data augmentation, regularization methods, an effective layout of the input layer, involving the fundamental motion characteristics, and of course the inherent ability of the CNN scheme in extracting high-level features. We conclude the shortage of a massive GPS dataset acts as the major deterrent for further progress in our CNN framework.

Table 2.7: Comparison of CNN's inference accuracy with five classical machine learning algorithms. HP: Hyperparameter, NA: Not applicable.

Model	**Test Accuracy (%)**
Seminal study Zheng et al.	76.2
Related study Endo et al.	67.9
Related study Wang et al.	74.1
Best model in this study (Ensemble of the model G)	**84.8**

2.6 Conclusion

In this article, we deployed the CNN architectures to infer transportation modes using only raw GPS trajectories. We established an unprecedented and state-of-the-art layout for the input layer of CNN architectures. We create a four-channel input volume, in which each channel represents a sequence of the basic kinematic attributes including speed, acceleration/deceleration, jerk, and bearing rate. We

applied several data preprocessing steps to eliminate anomalous GPS logs and random errors. Moreover, we divided the long segments into fixed length samples, which is a requirement in the CNN method. Since a good performance of the CNN is primarily contingent on access to the high number of samples, the subdivision process was very effective in enhancing the accuracy of the model by augmenting the number of samples more than four times. Afterward, we examined a set of CNN architectures with different layers' patterns so as to identify the most efficient one. We improved the prediction performance of the optimal CNN configuration by increasing the number of epochs in the training process. Using the early stopping method, the optimal number of epochs was selected so as to avoid overfitting problem. The reported metrics indicate that our ensemble of seven best CNNs not only performs well on its own way but significantly outperforms the classical machine learning algorithms, including KNN, SVM, DT, RF, and MLP. Furthermore, the test accuracy of our best CNN model exceeds the highest test accuracy reported by previous studies. It is worth noticing that removing anomalies, designing an efficient input layer with the appropriate motion characteristics, augmenting training data, tuning some hyperparameters, and employing the bagging concept are the key factors in attaining such high accuracy. According to our observation and analysis, the lack of access to a larger GPS-trajectory dataset was recognized as the leading reason for not achieving further improvement in our proposed CNN framework. As a future research direction, using the available unlabeled GPS trajectories and leveraging the semi-supervised and unsupervised learning algorithms can be a potential compensation for not accessing to a large labeled dataset.

Chapter 3

A Deep Learning Approach for Vehicle Classification Using Large-Scale GPS Data

Abstract

Transportation agencies are starting to leverage increasingly-available GPS trajectory data to support their analyses and decision making. While this type of mobility data adds significant value to various analyses, one challenge that persists is lack of information about the types of vehicles that performed the recorded trips, which clearly limits the value of trajectory data in transportation system analysis. To overcome this limitation of trajectory data, a deep **C**onvolutional **N**eural **N**etwork for **V**ehicle **C**lassification (**CNN-VC**) is proposed to identify the vehicle's class from its trajectory. Since a raw GPS trajectory does not convey meaningful information, this paper proposes a novel representation of GPS trajectories, which is not only compatible with deep-learning models, but also captures both vehicle-motion characteristics and roadway features. To this end, an open source navigation system is also exploited to obtain more accurate information on travel time and distance between GPS coordinates. Before delving into training the CNN-VC model, an efficient programmatic strategy is also designed to label large-scale GPS trajectories by means of vehicle information obtained through Virtual Weigh Station records. Our experimental results reveal that the proposed CNN-VC model consistently outperforms both classical machine learning algorithms and other deep learning baseline methods. From a practical perspective, the CNN-VC model allows us to label raw GPS trajectories with vehicle classes, thereby enriching the data and enabling more comprehensive transportation studies such as derivation of vehicle class-specific origin-destination tables that can be used for

planning.

Keywords: Inferring vehicle classes; Trajectory data; Vehicle classification; Virtual Weigh Stations

3.1 Introduction

The advances in Global Position Systems (GPS) and ever-increasing market penetration of GPS-equipped devices (e.g., smart phones) enabled generation of massive spatiotemporal trajectory data that capture human mobility patterns. Availability of large-scale trajectory data motivated transportation agencies to leverage such information for measuring transportation network performance and supporting their decision making. However, even though this type of mobility data adds significant value to various transportation analyses, one challenge that persists is lack of information about the classes of vehicles that performed the recorded trips. Namely, users of different GPS navigation apps (e.g., Google Maps) typically do not specify the type of vehicle they are driving, making it unclear to a transportation analyst whether a recorded trip was performed by a passenger car, small commercial vehicle, or even an eighteen-wheeler truck. Obviously, having this piece of information would be extremely useful for a wide range of applications in Intelligent Transportation Systems (ITS) including emission control, pavement designation, traffic flow estimation, and urban planning [54]. Furthermore, unlike fixed-point traffic flow sensors (e.g., loop detectors and video cameras), GPS sensors are not limited by sparse coverage and can be considered as an efficient and ubiquitous data source. Accordingly, our main objective in this paper is to enrich the increasingly available trajectory data by developing a model that is capable of identifying the type of a vehicle that produced a GPS trajectory.

However, the definition of vehicle categories is highly contingent on the ITS application and data-collection sensors. For example, vehicles can be classified based on the number of axles or weights for pavement designation and ETC systems. Among various categorizations, the vehicle-classification proposed by the US Federal Highway Administration (FHWA) in the mid 1980s has been accepted as the most standard scheme and served as the basis for a majority of state-counting efforts. The number of axles, the spacing between axles, the first axle weight, and

gross vehicle weight are the main factors for categorizing vehicles in the FHWA system. Table 3.1 provides information on the 13 vehicle categories according to the FHWA definitions. The FHWA vehicle classification might slightly change from State to State depending on the types of vehicles that are allowed to commute in that State. Nonetheless, in real-world situations, vehicles can be regrouped to more coarse-grained classes, such as {light, medium, heavy} [74], {passenger cars, trucks} [80], and {sedan, pickup truck, SUV/minivan} [43].

Table 3.1: FHWA Vehicle Classification System

#Class	Class Definition	#Class	Class Definition
1	Motorcycles	8	Four or Fewer Axle Single-Trailer Trucks
2	Passenger Cars	9	Five-Axle Single-Trailer Trucks
3	Other Two Axle, Four tires, Single-Unit	10	Six or More Axle Single-Trailer Trucks
4	Buses	11	Five or Fewer Axle Multi-Trailer Trucks
5	Two-Axle, Six-Tire, Single-Unit Trucks	12	Six-Axle Multi-Trailer Trucks
6	Three-Axle Single-Unit Trucks	13	Seven or More Axle Multi-Trailer Trucks
7	Four or More Axle Single-Unit Trucks	-	-

The common approach for classifying vehicles is based on fixed-point traffic sensors, which are categorized into two groups: 1) in-roadway sensors that are embedded in pavement or attached to road surfaces, including inductive-loop detectors, magnetometers, and Weigh-In-Motion (WIM) systems, 2) over-roadway sensors that are mounted above the surface, including microwave radar sensors, laser radar sensors, and ultrasonic infrared sensors. When a vehicle crosses them, these sensors are capable of calculating the vehicle's axle information (e.g., the number of axles and the space between them). Video image processor (VIP) is another over-roadway sensor that has extensively been deployed for the vehicle classification, in particular due to advances in vision-based techniques [36, 43]. After a vehicle is detected through VIP, computer-vision algorithms are exploited to first extract robust features from images and/or videos, and then classify the vehicle into predefined groups using supervised learning algorithms. However, there are several major shortcomings regarding the fixed-point traffic flow sensors including: 1) the installation and maintenance of such technologies not only requires extra costs but also necessitates road closure and physical changes to road infrastructure, 2) their performance is subjected to errors in various situations such as inclement weather conditions, high-speed road segments, and traffic congestion, 3) they can be used

only for the location where they were installed while many ITS applications require dynamic vehicle classification across a wide area.

One cost-effective way to address the above-mentioned issues is to use well-established positioning tools such as Global Position Systems (GPS) that enable to record spatiotemporal information of vehicles' trips. Unlike traffic flow sensors, GPS infrastructures does not impose extra costs as the system has already been designed for purposes not related to traffic management. Furthermore, GPS data are not limited by sparse coverage and can be considered as an efficient and ubiquitous data source for several transportation-domain applications including traffic state estimation, traffic incident detection, mobility pattern extraction, travel demand analysis, and transport mode inference [18]. Accordingly, this study seeks to harness the GPS data for addressing the vehicle-classification problem.

Before proceeding to development of a vehicle-classification model, for the first time, we label a large-scale GPS trajectory dataset according to the fine-grained FHWA vehicle categories. A vehicle's GPS trajectory is constructed by connecting a sequence of GPS points, recorded by means of an on board GPS-enabled device, where a GPS point contains information about the device's geographic location at a particular moment. The GPS data are recorded from vehicles that have traveled in the State of Maryland over the period of four months in 2015. To this end, a separate WIM dataset that contains the FHWA class information of the vehicles that passed Virtual Weight Stations (VWS) is used. An efficient programmatic-labeling strategy is designed to assign the vehicle class information from the WIM dataset to the GPS trajectories that have passed the corresponding VWS. Our programmatic labeling is accurate enough while saving costs and times compared to manual labeling, which is too expensive and time-consuming.

Having a large volume of labeled GPS trajectories, the GPS-based vehicle-classification problem can be examined for several vehicle-categorization scenarios. However, a raw GPS trajectory contains only a sequence of GPS points without any meaningful correlations and explicit features for feeding into supervised models. Feature engineering is common approach for extracting some hand-crafted features using the descriptive statistics of motion characteristics such as maximum speed and acceleration. Nonetheless, feature engineering not only requires expert knowledge but also involves biased engineering justification and vulnerability to traffic and

environmental conditions [17]. For example, different types of vehicles may have the same speed in a traffic jam. Automated feature learning methods such as deep learning architectures are a remedy for resolving shortcomings with hand-designed features. But, a raw GPS trajectory with a sequence of GPS coordinates and timestamps neither has an adaptable structure for passing into deep-learning algorithms nor conveys meaningful information. Accordingly, before deploying deep-learning algorithms, a novel GPS representation is designed to convert the raw GPS trajectory into a structure that contains both vehicle-motion characteristics and roadway-geometric information associated with the GPS trajectory. Afterward, a deep Convolutional Neural Network for Vehicle-Classification (CNN-VC) is proposed to identify the vehicle class from its GPS trajectory. The developed classification method can be subsequently deployed to identify vehicle classes of the remaining GPS trajectories in the original dataset, which adds a very important dimension to the dataset that is widely used to support analysis of the Maryland State Highway Administration (e.g., derive class-specific trip tables or link-based traffic volumes). Accordingly, this paper makes the following contributions:

- **Labeling a large-scale GPS trajectory dataset based on the FHWA classification scheme**. An efficient programmatic approach is developed to label GPS trajectories by means of vehicle class information obtained from VWS vehicle records. Information from an open source routing engine is also exploited to improve the accuracy of our labeling approach.

- **Designing a new representation for raw GPS trajectories**. First, a GPS trajectory is considered as a sequence of GPS legs, where each leg is the route segment between two consecutive GPS points. Then, a feature vector is computed for every GPS leg, which is a combination of motion-related and road-related features. Concatenating the feature vector corresponding to all GPS legs generates an efficient representation for each GPS trajectory.

- **Developing a deep Convolutional Neural Network for Vehicle-Classification (CNN-VC)**. For the first time in this domain, a CNN-based deep-learning model is developed to identify vehicle's class from the proposed GPS representation. The CNN-VC comprises a stack of CNN layers for extracting abstract features from the GPS representation, a pooling

operation for encapsulating the most important information, and a softmax layer for performing the classification task.

- **Conducting an extensive set of experiments for evaluating the proposed model**. In addition to the classical machine-learning techniques, several state-of-the-art deep-learning algorithms are developed to be used as baselines. Experimental results reveal that our CNN-VC model clearly outperforms both classical-supervised algorithms and deep-learning architectures.

The rest of this article is organized as follows. After reviewing related works in Section 3.2, the datasest and the labeling strategy are described in Section 5.3. The details of our proposed framework are elaborated in Section 4.4. In Section 4.5, numerous experiments are conducted to evaluate the performance of the proposed framework. Finally, conclusions are drawn in Section 4.6.

3.2 Related Works

Since vehicle classification is essential for many tasks in ITS, a considerable amount of literature has been published on developing frameworks for identifying the vehicle's class. Two primary types of sensors that have been used to develop vehicle-classification systems include: 1) fixed-point traffic flow sensors, and 2) floating sensors using GPS data. In this section, after briefly reviewing the literature on the first group, the few studies on the vehicle classification using GPS data are examined.

3.2.1 Fixed-point sensors for vehicle classification

A large and growing body of literature has addressed the vehicle-classification problem using various fixed-point traffic data-collection systems over the past decades. In-roadway (e.g., loop detectors, magnetic sensors, and piezoelectric sensors) and over-roadway (e.g., radar sensors, infrared sensors, and VIP) are the two main types of traffic flow sensors that have been exploited for classifying vehicles. Since in-roadway sensors interrupt traffic flow during installation and maintenance, they

are more appropriate for urban areas rather than high-speed roads such as freeways. On the other hand, over-roadway sensors are more appropriate for capturing vehicles information in high-speed, high-capacity roads as they are much less disruptive to traffic flows [80]. The idea behind vehicle classification using fixed-point sensors is to capture vehicles' physical characteristics (e.g., axle configuration and length) and then categorize vehicles according to a pre-defined set of classes. A dual-loop detector system, as an example of in-roadway sensors, can simply estimate the length of a vehicle by computing the speed and occupancy time and establish a length-based classification system [15]. A VIP system, as an example of on-roadway sensors, models vehicles as rectangular patches in a sequence of images recorded by a camera. Afterward, vehicles are classified to several groups using visual features (e.g., width, length, area, and perimeter) computed by traditional vision-based techniques [43] or abstract automatic features learned by deep-learning algorithms [20]. Sensors can also be integrated to form a more robust classifier system for fine-grained classification. A WIM system, which is comprised of piezoelectric sensors, VIP sensors, loop detectors, etc., is capable of classifying vehicles based on the 13-category FHWA classification scheme [37]. Further details on pros and cons of various vehicle-classification systems using fixed-point technologies are available in [80]. Since the specific objective of this study is to develop a GPS-based vehicle-classification system, we focus more on the studies that have harnessed GPS data for classifying vehicles.

3.2.2 GPS-based vehicle-classification

In spite of the popularity in fixed-point traffic flow sensors, they incur a high initial cost for installation and a permanent cost for maintenance. Furthermore, they are unable to spatially cover a wide area and in turn their usage is limited to the installation location. Probe vehicles equipped with positioning tools such as GPS can be a viable alternative to fixed-point sensors, due to the fast-growing market-penetration rate of GPS-enabled devices (e.g., smart phones). GPS trajectories, which summarize vehicle's spatiotemporal information, can be recorded when vehicles are performing regular trips. Over the past decade, much research has been carried out to harness GPS trajectories for several mobility applications, ranging from vehicle fleet management, human mobility behavior, and inferring

significant locations to travel mode detection and travel anomaly detection [18]. A comprehensive and systematic review on various aspects of trajectory data mining including trajectory data preprocessing, trajectory data management, and several trajectory mining tasks is available in [96]. Although numerous studies leveraged GPS data in various mobility applications, only a few studies have developed vehicle-classification systems based on GPS trajectories.

[80], for the first time, utilized GPS data to distinguish passenger cars from trucks, yet on a very small dataset with 52 samples of passenger cars and 84 samples of trucks. Compared to speed-related features, they found that the features related only to acceleration and deceleration (e.g., the proportions of acceleration and deceleration larger than 1 m/s^2, and the standard deviations of acceleration and deceleration) result in a better classification performance. Acceleration-based features are then passed into the Support Vector Machine (SVM) classifier for the classification task. In a recent study by [74], a deep-learning framework based on Recurrent Neural Networks (RNN) was developed to classify vehicles into three categories: light-duty, mid-duty, and heavy-duty. The input data for the RNN model is a sequential point-wise GPS information. Such a sequence is created by computing a set of point-wise features including distance, time, speed, acceleration, and road type for every GPS point in a GPS track. Their proposed model was trained using almost 1 million GPS tracks collected by a fleet intelligence company. Note that a growing body of literature has recently leveraged the power of deep learning algorithms for solving various applications in the transportation domain [16, 72].

A similar research track belongs to travel mode detection using GPS trajectories. There the objective is to identify transportation mode(s) used by travelers for making their regular trips based on GPS data. Analogous to the vehicle-classification problem, the problem of travel mode detection consists of two main steps: (1) extracting features from GPS logs, either by means of feature engineering [97] or automated feature learning methods [17], (2) feeding features to learning algorithms for the classification task. However, it is obvious that distinguishing between the GPS patterns of transportation modes (e.g., walk and car) is less challenging than discriminating between GPS patterns of vehicles (e.g., passenger car and pickups).

To the best of our knowledge, no GPS dataset annotated with the FHWA vehi-

cle categories is available. Thus, we first design an efficient time-based-matching process for labeling a large number of GPS trajectories by the help of a separate WIM dataset, which contains the FHWA class information. Furthermore, a novel representation for GPS tracks is proposed to consider both kinematic motion characteristics of vehicles and roadway features for all road segments between every two consecutive GPS points in a GPS track. Finally, for the first time, a convolutional-based deep-learning model is applied to the new GPS representation for discriminating GPS trajectories according to their vehicle classes.

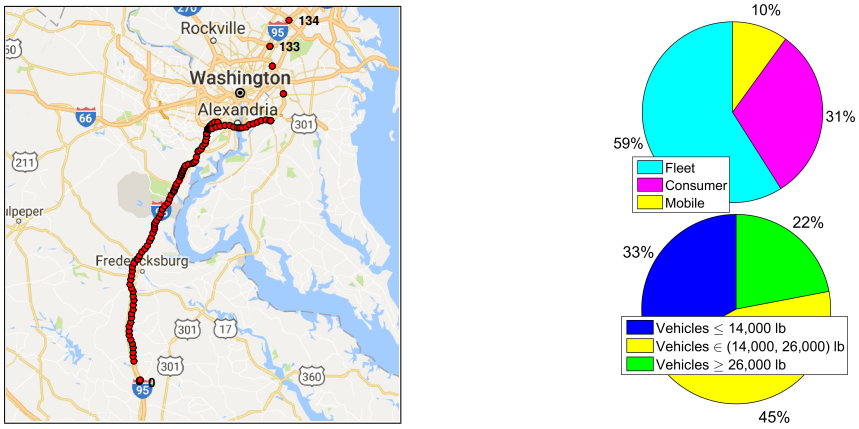
3.3 Datasets description and labeling strategy

The main purpose of this section is to develop an efficient scheme for labeling a large-scale GPS trajectory dataset based on the FHWA classification scheme. To the best of our knowledge, no GPS dataset has yet been labeled according to a fine-grained vehicle-class system. Such valuable labeled dataset can be exploited in a variety of trajectory mining and mobility studies besides the vehicle-classification problem. For example, knowing vehicle classes for all the recorded trips would enable derivation of trip tables for different vehicle classes (e.g, separate for passenger and commercial vehicles) and estimation of not only aggregate link-based volumes but their vehicle compositions.

However, manual labeling of a large-scale dataset is obviously an expensive and time-consuming task, which calls for a programmatic labeling approach. To this end, vehicle class information obtained from VWS records is deployed as an auxiliary dataset for labeling GPS trajectories. Information from an open source routing engine is also extracted to maximize the quality of the labeling process. Using these resources, an efficient programmatic approach for labeling GPS trajectories is proposed to not only dramatically save time and cost but also ensure accurate labeling of GPS data. In this section, we first describe GPS trajectory data, VWS data, and Open Source Routing Machine (OSRM), and then elaborate on the labeling approach. Lastly, the distribution of labeled GPS data among vehicle classes, obtained by applying our labeling strategy to 20 million GPS trajectories, is provided.

3.3.1 Data sources description

3.3.1.1 GPS trajectory Trajectory data used in this paper comes from one of the leading GPS data providers in North America, which collects data from several hundred million vehicles and devices across the globe. As a sample data, Fig. 3.1a presents a single GPS trajectory, which consists of vehicle locations and corresponding timestamps. The considered dataset includes 20 million GPS trajectories over a period of four months in 2015: February, June, July, and October. Every GPS trajectory is defined by a unique trip ID and a device ID. Trajectories have also been categorized into three groups based on the vehicle gross weight: (1) below 14,000 lbs (FHWA classes 1-3), (2) between 14,000 lbs and 26,000 lbs (FHWA classes 4-6), and (3) above 26,000 lbs (FHWA classes 7-12). These trajectories span over the entire state of Maryland and include a relatively high percentage of vehicles with weights above 14,000 lbs, as shown in Figure 3.1b.



(a) Sample trajectory (b) Trip info

Figure 3.1: A sample GPS trajectory and info about 20 million trips (i.e., vehicle weights and data providers).

3.3.1.2 Virtual Weight Stations Data A VWS is a roadside enforcement facility that eliminates the need for continuous staffing by automatically identifying and recording vehicles' characteristics on a real-time basis. Each station is comprised of several components, including WIM sensors (for computing vehicles' weights and axle configuration), camera system (for real-time identification of vehicles), screening software (for integrating data from WIM and camera systems), and

communication infrastructure (for transferring the recorded information to authorized personnel such as mobile enforcement teams). Accordingly, WIM sensors in a VWS are capable of determining the approximate gross weight and axle configurations (e.g., weight, number, and spacing) in real-time, which are in turn used for the vehicle classification [10].

The VWS records considered in this study include information collected at 7 VWS in the state of Maryland over the period of four months in 2015: February, June, July, and October. However, due to programmatic nature of the labeling strategy described in the following sections, only three stations installed on road segments with one lane per direction are utilized for the labeling purpose. Figure 3.2 illustrates the approximate location and the roadway network around these three VWS. For every vehicle crossing WIM sensors, various information has been recorded including: crossing time, crossing lane, class, number of axles, speed, gross weight, and length. The class definitions are based on the FHWA vehicle-classification system, as described in Table 3.1. Class 1 and 13 are not available in this dataset. Among all attributes, only vehicle's crossing time, class, and gross-weight features are deployed in our labeling process. It should be noted that there is no pre-relation between the vehicles in GPS and VWS datasests. Indeed, the main goal of our labeling strategy is to match vehicles in these two datasets.

3.3.1.3 Open Source Routing Machine Open Source Routing Machine (OSRM) is a web-based navigation system that computes the fastest path between an origin-destination pair, using the Open Street Map (OSM) data. The other functional services that OSRM provides, include map-matching, nearest matching, traveling salesman problem solving, and generating vector tiles. All of these services are accessible through its Application Programming Interface (API) methods. The main advantage of the OSRM APIs is that it can be used for free and without usage limits, which makes it a valuable resource for research purposes. The following OSRM services are exploited throughout this study for both labeling process and the vehicle-classification system:

- *Map-matching*: Given a set of GPS points (e.g, a GPS trajectory), the map-matching is the service that snaps the GPS points to the OSM network in the most plausible way. Attaching GPS points to the most likely nodes in the

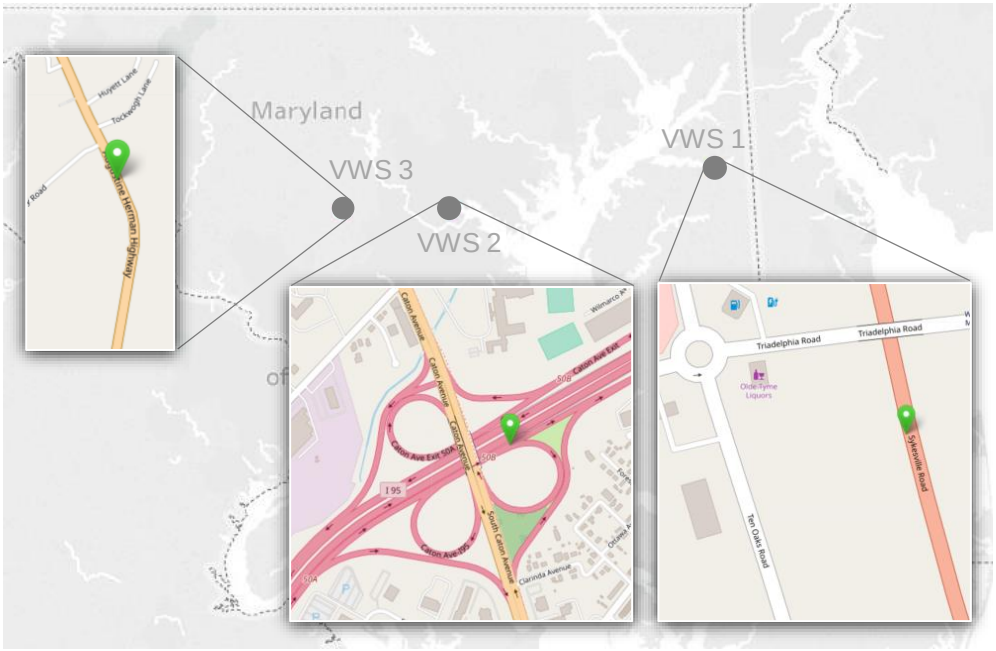


Figure 3.2: The location and roadway network associated with three Virtual Weight Stations (VWS) in the state of Marlyland, used for the labeling process. The green map markers depict the location of VWS.

road network alleviates location errors associated with the GPS technology. Erroneous GPS logs, that cannot be matched successfully, are considered as outliers and discarded. When map-matching is completed, a variety of fine-grained information between every two consecutive matched points are provided including travel distance, travel time, number of turn movements, number of intersections, etc.

- *Nearest*: The nearest service snaps a GPS coordinate to the nearest location in the traffic network. This service is particularly useful when the accurate location of a GPS log, rather than a GPS trajectory, matters. The name of street, to that the GPS log is snapped, is the most useful information provided by the nearest service.

- *Route*: The route service finds the fastest route between two GPS coordinates. Although the map-matching service can provide the similar information, the route service is a more optimized way when the goal is to compute travel time and distance for only one origin-destination pair.

3.3.2 Labeling GPS trajectories

First, a vehicle's GPS trajectory is defined as follows, which is used throughout this study:

Definition 1 (Vehicle's GPS trajectory). *A vehicle's GPS trajectory T is defined 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.*

The key idea behind our proposed labeling process is to compute the crossing time-window of a vehicle's GPS trajectory from a VWS and then identify its class among those vehicles that crossed the station at the computed time-window. Thus, our proposed labeling strategy consists of the following five major steps that should be taken in sequence:

1. **Retrieving initial GPS trajectories.** GPS trajectories that have potentially crossed one of the three stations are filtered out from the whole GPS dataset. This eliminates the need of involving a large volume of GPS trajectories that obviously have not crossed any of VWS and in turn there is no chance to be labeled. Temporarily removing these GPS trajectories from our labeling pipeline significantly reduces the computation time in the next steps.
2. **Applying filtering criteria.** Every GPS trajectory, obtained from the first step, is re-examined through several filtering rounds to guarantee that the GPS trajectory has truly crossed one of the three VWS. The GPS trajectories that have crossed a VWS and their corresponding VWS are determined and used in the next step.
3. **Predicting crossing time-window.** For every identified trajectory in the previous step, the time-window that the GPS trajectory has crossed the VWS is predicted.
4. **Labeling GPS trajectories.** The class information of all vehicles that have crossed the VWS during the predicted crossing time-window are retrieved. The GPS trajectory is labeled only if all crossing vehicles have the same class.

5. **Augmentation of labeled data.** Trajectories with the same device ID have been operated with the same vehicle. Hence, after labeling a portion of trajectories retrieved from the first step, trajectories with the same device ID as the trajectories labeled in the previous step are fetched from the database containing all 20 million GPS traces and labeled accordingly.

The above four steps are explained in the following sections.

3.3.2.1 Step 1: Retrieving initial GPS trajectories Using the OSM API, only trajectories that include a GPS point within a two mile radius from the 3 VWS are queried for the subsequent analysis. This reduces the number of trajectories from the initial 20 million to about 300,000 trajectories. The rationale behind this quick step is to dramatically expedite the processing time in steps 2 to 4, as the most computationally-intensive steps in our labeling pipeline.

3.3.2.2 Step 2: Applying filtering criteria The retrieved trajectories from step 1 have not necessarily passed one of the three stations due to several reasons. For example, both directions of a roadway in OSM are often represented by one 'way' component, in which 'way' is one of the basic elements in OSM that are used to represent linear features such as roads. Therefore, a trajectory found in the first step might have a reverse direction with respect to the VWS, as shown in Figure 3.3a. Also, some of the retrieved trajectories are only close to a VWS without passing the station. Accordingly, we implement several strong filtering criteria to ensure a GPS trajectory has crossed the station. The possibility of a GPS trajectory crossing a station is examined one-by-one for all three stations. Since stations are far apart, we assume that a GPS trajectory can cross only one station within a few hours. Thus, if a GPS trajectory passes all criteria for the first station, the remaining stations will not be examined for that particular trajectory. Finally, our filtering criteria in this step have been designed to simultaneously obtain the GPS points of a trajectory that are adjacent to the crossed VWS, which is an essential information for the next step (i.e., predicting and matching the crossing time-window). This is another way to improve the overall efficiency of our labeling scheme. Before proceeding to the filtering criteria, the following notation is described for an easier understanding of the filtering approach.

Definition 2 (Before/after GPS point lists). *Let a VWS be represented by S. The before/after GPS point lists of the trajectory T with respect to the station S, denoted as BL_{TS}/AL_{TS}, is defined as all GPS points in T in which their timestamp attribute are less/greater than the approximate time that the GPS trajectory T might have passed the station S. This crossing time is denoted as C_{TS}. Note that the before/after lists are defined based on the time, not the location. BL_{TS}/AL_{TS} are mathematically defined as follows:*

$$BL_{TS} = \{p[t] \leq C_{TS} \mid p \in T \text{ and the station is } S\}$$
$$AL_{TS} = \{p[t] \geq C_{TS} \mid p \in T \text{ and the station is } S\}$$

Every GPS trajectory T is looped over the three VWS and, for every VWS, the following filtering criteria are examined in sequence. If T fails to pass a criterion, T is discarded for labeling and the remaining criteria are not examined. Thus, the GPS trajectory T crosses a VWS S only if it passes all the filtering criteria.

1. The GPS trajectory T needs to have at least one GPS point located before the station S in order to cross the station. For example, since the station 1 is installed on the southbound, the GPS trajectory T is crossing the station 1 only if a GPS point with the latitude greater than the latitude of S is found in T. If any GPS point found, then the closest GPS point to the S is considered as the before-GPS-point. Note that, since the trip direction of a vehicle might change several times along the trip path, not all GPS points geographically located before the station are considered in the before list BL_{TS}. The GPS points that are recorded in a time order before the before-GPS-point form BL_{TS}.

2. At least one GPS point needs to exist after the before-GPS-point and located after the station S in the GPS trip T; otherwise, T has not crossed the station S. If exists, this GPS point is considered as the after-GPS-point. An example of a GPS trajectory that does not have an after-GPS-point (i.e., has not crossed the station) is depicted in Figure 3.3b .

3. The distance of before-GPS-point and after-GPS-point to the station S must be less than a specified threshold; otherwise, T has not crossed the station

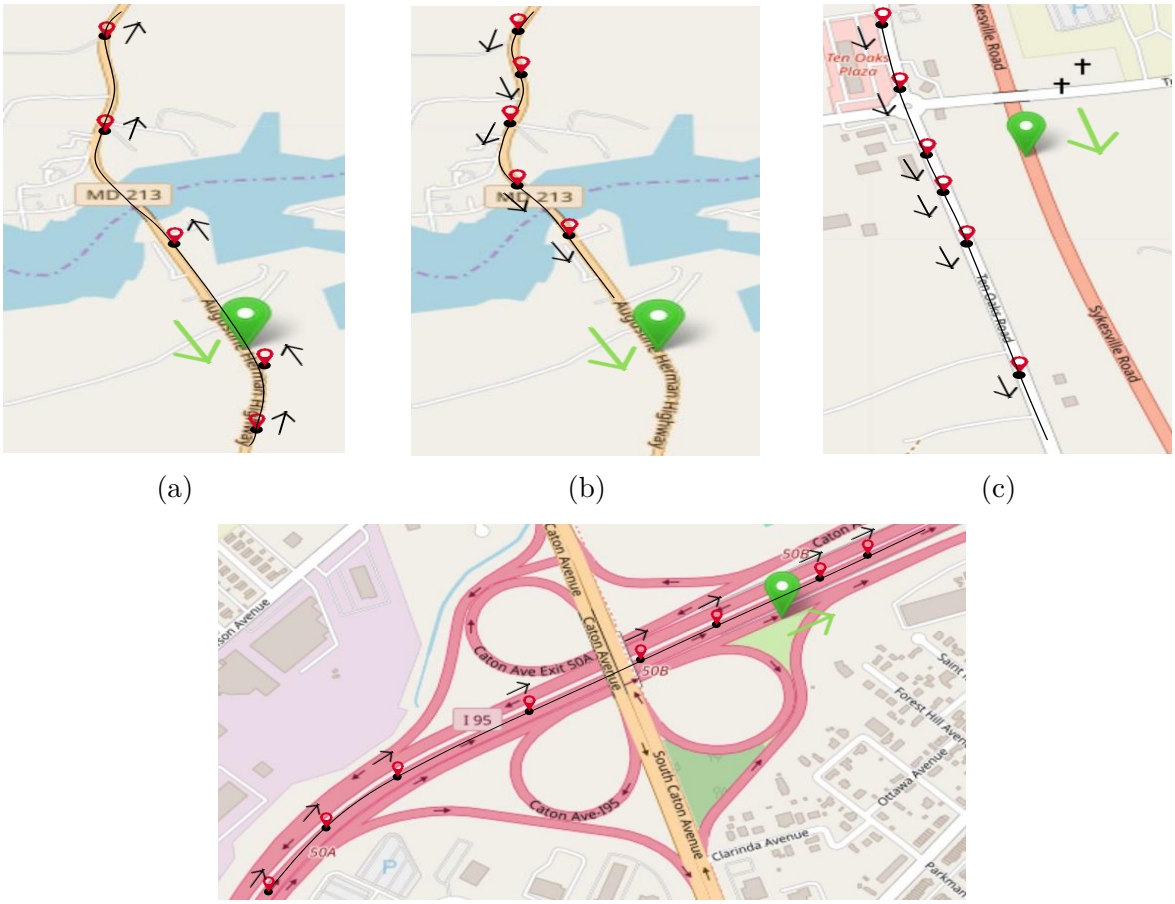
S. For example, it might pass a street parallel to the station S, as shown in Figure 3.3c. The threshold value in our labeling process is set to 2 miles. Note that if a GPS trajectory is passing through a parallel street yet its distance to the station is less than the specified threshold, the GPS trajectory is detected and discarded by the next filtering criterion.

4. Due to the existence of parallel and similar roadways around a station, there is still a chance that a GPS trajectory passes all the previous criteria without actually crossing the station. This situation particularly happens for the station 2, in which the road geometry around this station is very complex with several grade-separated road junctions. This complex interchange contains road sections that are parallel with the VWS lane. This causes to detect many GPS trajectories that have passed the previous criteria while they have not crossed the station 2, as shown in Figure 3.3d. Accordingly, more restricted criteria are implemented. In this case, we utilize the nearest service in OSRM to snap the GPS points of T, that are close to a VWS, into the nearest coordinates in the traffic network. The names of roads associated to each snapped GPS point are then retrieved. The GPS trajectory T has crossed the VWS only if it has traversed the road sections that end up to the VWS location.

3.3.2.3 Step 4: Predicting time-window We have thus far obtained some GPS trajectories that have crossed one of the three VWS. The before and after GPS point lists have also been identified for each GPS trajectory with respect to its corresponding station. Having the GPS trajectory T and the crossed station S, the following steps are performed to approximate the time-window that T has crossed S. Figure 3.4 illustrates the process of predicting the time-window for a sample GPS trajectory.

1. *Computing the approximate crossing time.* Using the OSRM route service, the fastest route between the before-GPS-point (or alternatively the after-GPS-point, whichever is closer to the station S) and the station S is computed. The approximate crossing time of station S by the GPS trajectory T is then computed as follows:

$$C_{TS} = p_b[t] + TR_{osrm} \textbf{ OR } C_{TS} = p_a[t] - TR_{orsm} \tag{3.1}$$



(a) (b) (c)

(d)

Figure 3.3: Examples of GPS trajectories that have failed to pass one of the filtering criteria. The green and red markers indicate the VWS and GPS point locations, respectively. The green and black arrows indicate the road direction associated with the VWS and GPS trajectory, respectively. (a) GPS trajectory is moving in a reverse direction with respect to the VWS. (b) GPS trajectory is moving along a parallel road with respect to the VWS. (c) GPS trajectory approaches to the VWS, yet not crossing. (d) GPS trajectory is moving along an alternative route, rather than the VWS road.

where $p_b[t]$ and $p_a[t]$ are the timestamps for the before-GPS-point and after-GPS-point, respectively. TR_{orsm} is the travel time duration between either $p_b[t]$ or $p_a[t]$ to the station S, retrieved from the OSRM route service.

2. **Computing the travel-time error.** Travel time retrieved from OSRM does not exactly match with the real travel time recorded by GPS-enabled devices. Thus, we need to obtain the travel-time error introduced by OSRM,

which is then used as the time-window around C_{TS}. Two immediate before and after GPS points from the BL_{TS} and AL_{TS} are selected and concatenated to form a GPS sequence for the time-window prediction. After snapping the GPS sequence into the traffic network using the map-matching service, the travel times computed by OSRM between every two consecutive GPS points in the GPS sequence are obtained. Comparing OSRM travel times against real travel times computed from the timestamp attribute of GPS points results in the average error as follows:

$$MAE = \frac{\sum_{i=1}^{N} \mid TR_{i-real} - TR_{i-osrm} \mid}{N} \tag{3.2}$$

where MAE is the mean absolute error in approximating C_{TS}. In other words, MAE shows the error imposed by OSRM when computing C_{TS} according to Equation (3.1). In computing the travel-time error, we particularly use very few points (i.e., two immediate GPS points from the before and after lists) around the VWS to consider traffic conditions and driver behavior only around that specific location, rather than over the whole GPS trajectory.

3. **Computing crossing time-window.** Finally a crossing time-window is constructed around the C_{TS} using the average travel-time error as follows:

$$C_{TS} - MAE \leq C_{TS} \leq C_{TS} + MAE \tag{3.3}$$

3.3.2.4 Step 5: Labeling GPS trajectories After predicting the time window for C_{TS}, we assign the class label to the T according to the following steps:

1. As mentioned, the raw GPS data have categorized into three weight groups by the GPS provider. The vehicle records that their crossing time and gross weight attributes fall in the predicted time-window and the original weight category, respectively, are fetched from the VWS vehicle records associated with the station S. Note that no vehicle records might be retrieved from the VWS dataset as the predicted crossing time-window is commonly very short due to the low amount of error in Equation (3.5). Low travel-time error is a good sign about our crossing-time prediction accuracy.

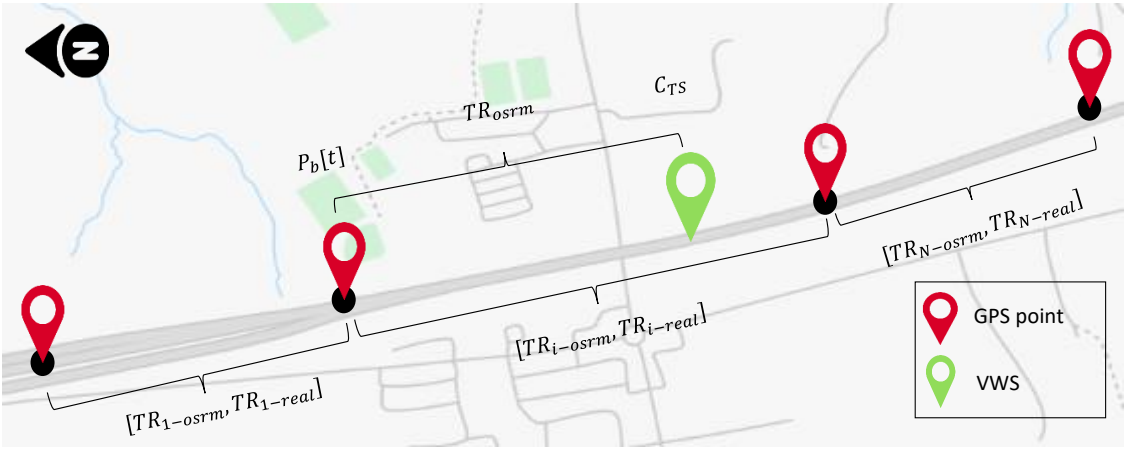


Figure 3.4: Predicting the crossing time-window from a VWS for a GPS trajectory. Notations are referred inside the text.

2. The GPS trajectory T is labeled to a vehicle class **only if** all vehicle records retrieved in the last step are the same. Although this leads to a smaller number of labeled GPS trajectories, such a conservative approach results in very accurate labeling. The GPS trajectory is discarded as an unlabeled trajectory if a unique class is not found among all retrieved vehicle records. The 'Before Augmentation' column in Table 3.2 shows the number of GPS trajectories that have been labeled using our labeling approach. Only 6,906 trajectories out of almost 300,000 initial GPS trajectories, retrieved from the first step of the labeling approach, have been labeled through our proposed strategy. The low number of labeled data at this stage, which mainly stems from applying strong constraints, is a good sign that our approach only labels GPS trajectories that have truly crossed a VWS.

3.3.2.5 Step 6: Augmentation of labeled data Finally, in order to augment labeled data, the GPS trajectories having the same device ID as the trajectories labeled in the previous step are extracted from the dataset containing 20 million trajectories and annotated with corresponding vehicle classes. As can be seen in Table 3.2, extracting trajectories with the same device ID significantly augments the number of labeled trajectories, from almost 7,000 to 500,000. Due to the augmentation opportunity, we deliberately implemented strong filtering criteria and labeling constraints so as to obtain labeled GPS data with the highest possible quality out of a programmatic approach. Moreover, due to the lack of ground

truth data, the proposed strategy was refined through several iterative visualizations to ensure the accuracy of labeled data. It should be noted that the 'After Augmentation' column in Table 3.2 summarizes the output of our labeling process while the 'After Processing (Final)' column shows the number of labeled GPS data per class used for developing CNN-VC after applying the pre-processing steps described in Section 4.5.

Table 3.2: Number of labeled GPS trajectories per vehicle class obtained in various stages

Vehicle Class	Before Augmentation	After Augmentation	After Processing (Final)
Class 2	3,306	83,917	42,473
Class 3	46	8,819	5,961
Class 4	86	12,240	8,128
Class 5	1,372	302,960	100,000
Class 6	208	40,361	24,610
Class 7	356	43,717	29,178
Class 8	100	13,006	7,968
Class 9	630	17,516	10,777
Class 10	16	105	85
Class 11	3	20	16
Class 12	1	689	284
Total	6,906	523,350	229,480

3.3.2.6 Scalabitlty of the labeling approach As described, our proposed labeling approach is a computationally-extensive task since labeling every GPS trajectory demands numerous computation steps. This makes the labeling of such a large-scale GPS data unscalable without using parallel computing systems. Apache Spark, a distributed computing engine that processes data in parallel, is deployed to make this procedure scalable by expediting the process 13x faster. For instance, the computation time for labeling one month of GPS data reduced from 8 hours to 35 minutes by means of Apache Spark.

3.4 The Proposed Model

The proposed CNN-VC is comprised of two main components: 1) New representation for GPS trajectories, and 2) Convolutional Neural Network (CNN). In

this section, these two components and the way they interact with each other for identifying vehicles' classes from GPS trajectories are elaborated. The process for training our CNN-VC is also described in this section.

3.4.1 GPS trajectory representation

As defined, a GPS trajectory contains only a series of chronologically ordered points, with only coordinate and timestamp information for each point. The raw values in such an ordered list is incapable of representing the vehicle's moving pattern and roadway characteristics, because the high or low values of coordinates and timestamps do not convey any meaningful information. Moreover, although this numerical ordered list can be blindly fed into traditional supervised algorithms (e.g., SVM), its structure is not adaptable for deep-learning models such as CNN and RNN. Therefore, a novel representation of GPS trajectories suitable for deep learning algorithms is proposed, which accounts for both the vehicle's motion features and roadway characteristics.

Unlike other studies [17, 74], we leverage the OSRM APIs to obtain more accurate motion information and access to fine-grained roadway features. First, GPS coordinates in a raw GPS trajectory are snapped into the traffic network using the OSRM map-matching service. The matching process is accompanied by several advantages: 1) Every GPS point is snapped to the nearest node in the traffic network, which in turn alleviates errors that may be attributed to the GPS technology (e.g., satellite orbits, receiver clocks, and atmosphere effects) and the GPS data provider (i.e., rounding the latitude/longitude decimal numbers to the 4th decimal number), 2) The outlier GPS points are removed if they cannot be appropriately matched, 3) The sequence of matched points represents a real and plausible route in the network. The matched GPS trajectory can be imagined as a sequence of GPS legs, denoted as $lg \in T$, $T = [lg_1, \ldots, lg_N]$, where every lg is the route segment between two consecutive matched GPS points. Such a structure is analogous to the sentence structure in Natural Language Processing (NLP) tasks, in which a sentence is comprised of a word sequence. Our main objective in this section is to represent every lg with a numerical feature vector. The feature vector corresponding to a GPS lg posses two types of features: motion-related and road-related

features.

The following motion features are computed for every lg in a GPS trajectory T:

- *Distance*, denoted as D: Distance is the accurate map-based length traveled by lg, which is extracted from the OSRM API. Unlike other studies [17, 74], we do not compute the geodesic distance (i.e., the direct distance between two locations) using well-known formulas such as Haversine and Vincenty. Instead, the map-based distance is computed using the OSRM API to take the traffic network configuration into account.
- *Duration*, denoted as Δt: Duration is the travel time for lg. The travel time is calculated based on the original timestamp information of GPS points, rather than the travel time computed by OSRM.
- *Speed*, denoted as S: Speed is the rate of change in distance for traveling the road segment of lg that shows how fast a driver is moving. Speed is calculated based on the map-based distance D and the real duration Δt.
- *Acceleration*, denoted as S: Acceleration is the rate of change in speed. The acceleration for the current lg is calculated based on the speed change of the current and subsequent legs. Accordingly, the last lg in T will be discarded.
- *Bearing rate*, denoted as BR. Bearing rate for lg indicates to what extent the vehicle's direction has changed between the start and the end points in lg. BR is the difference between bearings of the starting and end points. The bearings of start/end points are the clockwise angle from true north to the direction of the travel immediately after/before the start/end points of the lg. Likewise to distance D, the bearing values are retrieved from OSRM so as to obtain more accurate information. Note that bearing rate is an important motion feature that varies among vehicles [17, 97]. For example, a semi-trailer may not travel in a route segment that requires a sharp change in the heading direction.

The overall characteristics of the roadway that a vehicle uses for performing a trip is a discerning factor for developing a robust vehicle-classification system. It is apparent that various types of vehicles (e.g., passenger cars versus trucks)

have different transport-infrastructure preferences for making their regular trips. Furthermore, vehicle regulations may ban some vehicles (e.g., semi-trailer trucks) from traversing in dense urban environment. Accordingly, the following roadway characteristics are extracted from the OSRM API for every lg in a GPS trajectory T.

- *Number of intersections*, denoted as IN. Intersection is defined as any cross-path the vehicle passes when traveling along lg. IS is the number of all intersections in lg.
- *Number of maneuvers*, denoted as M. Maneuver is defined as any type of movement change that a vehicle needs to take when traversing lg. Turning right/left, traversing roundabout, taking a ramp to enter a highway, and merging onto a street are the examples of maneuver.
- *Road type*, denoted as R. Although a wide range of schemes are available for classifying roadways, a coarse-grained scheme is selected to divide roads into three groups: 1) Low-capacity roads including alley, street, avenue, boulevard, etc. 2) High-capacity roads including arterial, expressway, turnpike, parkway. 3) Limited-access grade-separated roads including freeway, highway, motorway, beltway, thruway, bypass, etc. The road type associated to each GPS lg can be extracted using the OSRM nearest service. Since R is represented as one-hot encoding, a coarse-grained road-type scheme avoids having a high-dimensional sparse feature vector.

The feature vector corresponding to every GPS lg is then created by concatenating the above-described motion and road features. Let $x_i \in \mathbb{R}^d$ represent the feature vector corresponding to the i-th leg in a trajectory, where d is the feature-vector dimension. Horizontally concatenating feature vectors of all legs in the GPS trajectory results in the new matrix representation, denoted as $\mathbf{X} \in \mathbb{R}^{L \times d}$, where L is the number of legs in the GPS trajectory. The structure of $\mathbf{X} \in \mathbb{R}^{L \times d}$ is shown in Figure 3.5 and is used as the input matrix for deep-learning models. Since deep learning requires $\mathbf{X}$ for all samples in a training batch to have a fixed size, we either truncate long GPS trajectories or pad short ones with zero values into the fixed size L. Our observation indicates that setting L to a high percentile value (i.e.,

a value between 75-85%) of the number of legs in all GPS trajectories improves the overall model performance as long as trajectories can be represented by more motion and roadway features.

3.4.2 Convolutional Neural Network

CNN is a sophisticated network that is capable of capturing local correlations in spatial structures. In each convolutional layer of CNN, local correlations between adjacent input values are perceived by convolving a filter across the whole surface of an input volume. Considering $\mathbf{X}$ as a spatial representation, CNNs can obtain the correlation between consecutive GPS legs by convolving their corresponding feature vectors. The output of the convolution operation performed by each filter is called a feature map. Concatenating feature maps corresponding to multiple filters results in a new volume, which is an abstract representation for the GPS trajectory. The subsequent convolution layer will extract a more abstract representation of the output volume from the previous layer.

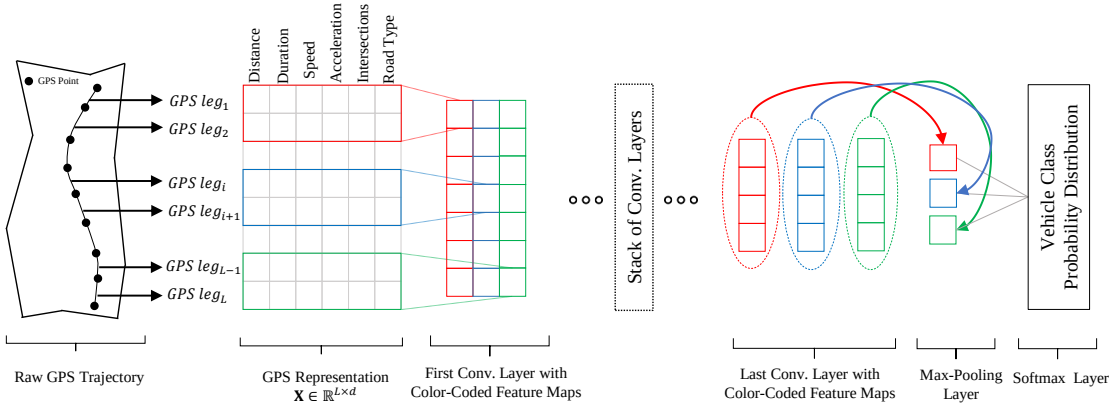


Figure 3.5: The structure of the proposed GPS representation and CNN-VC model. The CNN-VC comprises a stack of convolutional layers followed by max-pooling and softmax layers. Each color code in the convolutional layer is corresponding to one filter and its feature map

Figure 3.5 depicts our proposed architecture for identifying the vehicle class associated with a GPS trajectory. As can be seen in Figure 3.5, a vehicle's class is detected by passing its GPS trajectory $\mathbf{X}$ through multiple sets of convolutionsal layers that are stacked together. The convolution operation in each layer involves

a filter $F \in \mathbb{R}^{n \times d}$, where n is the number of consecutive GPS legs in the GPS trajectory and d is the filter width equal to the leg's feature-vector dimension. It is very important to set the width of the filter equal to the feature-vector dimension since the principal goal is to extract spatial correlation between consecutive legs. Sliding the filter F across the matrix $\mathbf{X}$ produces a feature map $\mathbf{h} \in \mathbb{R}^{(L-n+1)\times 1}$, where each element h_i of the feature map is computed as

$$h_i = f(F \circ X_{i:i+n-1} + b), \tag{3.4}$$

where $\circ$ is the dot product between the parameters of filter F and the entries of submatrix $\mathbf{X}_{i:i+n-1}$, b is a bias term, and f is a non-linear activation function. Rectified Linear Units (ReLU) is utilized as the non-linear activation function f throughout this paper. $\mathbf{X}_{i:i+n-1}$ refers to concatenation of n consecutive GPS legs at position i in the GPS trajectory. Using K filters of the same size as F, K feature maps are generated. Vertical concatenation of the created feature maps results in a new GPS representation matrix $\mathbf{X}_{new} \in \mathbb{R}^{(L-n+1)\times K}$, formally defined as

$$\mathbf{X}_{new} = \big[\mathbf{h}_1, \cdots, \mathbf{h}_{L-n+1}\big]. \tag{3.5}$$

The i-th row of the matrix $\mathbf{X}_{new}$ is an abstract feature vector for the n consecutive legs in the GPS trajectory. $\mathbf{X}_{new} \in \mathbb{R}^{(L-n+1)\times K}$ is used as the input volume for the next convolutional layer, where the same convolutional operation is applied. Our network can comprise several sets of convolutional layers, where every set has two convolutional layers with the same filter size and number of filters (i.e., n and k). However, the filter size and the number of filters may vary among layer sets. Finally, a max-pooling layer is applied to each feature map of the last convolutional layer. The rationale behind applying the max-pooling layer is to take one feature with the highest value (probably as the most important one) from each feature map. Such a max-pooling operation generates a feature vector that summarizes the high-level, abstract representation of a GPS trajectory. As the last step, the final vector representation is directly passed into the softmax function to perform the classification task by generating a probability distribution over vehicle classes for each GPS trajectory T, denoted as $P = \{p_1, \cdots, p_C\}$, where C is the number of vehicle classes. It should be noted that no fully-connected layers are used between the convolutional layers and the softmax layer.

3.4.2.1 Training and Regularization Our proposed architecture is trained by minimizing the categorical cross-entropy as the loss function, which is formulated for every GPS trajectory T as

$$\mathcal{L}(\theta) = -\sum_{i=1}^{H} y_{T_i} \log(p_{T_i}), \tag{3.6}$$

where θ are all learnable parameters in the model. $y_i \in \mathbf{Y}$ is a binary indicator which is equal to 1 if the class i is the true vehicle class for the GPS trajectory T and 0, otherwise. $\mathbf{Y}$ is the true label for T, represented as one-hot encoding. The cross-entropy loss is averaged across the training batch in each iteration. The Adam optimizer is used to update model parameters in the back-propagation process. Adam is a well-suited optimization technique for problems with large dataset and parameters that has recently seen broader adoption for deep learning applications [46]. We use Adam's default settings as provided in the aforementioned paper: $learningrate = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The network weights are initialized by following the scheme proposed in [30].

The vehicle-classification problem often suffers from an imbalanced distribution among vehicle classes, where the number of samples belonging to each class does not constitute an equal portion of the training dataset. Training an imbalanced dataset in the same way as a balanced one and without implementing appropriate strategies results in a poor classifier that is not robust against non-dominant classes. Therefore, the following strategies are implemented in our training process in order to achieve a robust classifier:

- *Random over-sampling.* The number of GPS trajectories in all minority classes increases to the number of GPS trajectories in the majority class by randomly replicating them.
- *Balancing training batch.* Equal proportion of GPS trajectories from each class forms the training batch in each training iteration.
- *Cost-sensitive learning.* Higher weights are assigned to minority classes in the cross-entropy loss, Equation (4.9), where weights are determined according to their class proportion in the dataset. This strategy penalizes misclassifications of the minority classes more heavily than the majority classes.

It is worth noting that the random over-sampling and balancing the training batch significantly improves our model performance while the cost-sensitive learning has a low impact.

Dropout and early-stopping are two types of regularization that are applied to our architecture. The dropout layer is added before the last layer (i.e., the softmax layer) with setting the dropout ratio to 0.5. In the early-stopping methods, the training is stopped if the performance metric does not improve on the validation set after two consecutive epochs. The model with the highest validation score is restored for applying on the test set.

3.5 Experimental Results

In this section, first, several variants of the labeled GPS data are created. Afterward, the performance of our proposed model will be evaluated on the created datasest by comparing against several classical and deep-learning models. Robust classification metrics are used to assess the prediction quality of our proposed model. The quality of our proposed GPS representation is also investigated. Finally, the overall quality of the proposed model for predicting different types of vehicle categorization is discussed.

3.5.1 Datasets definitions and preparations

The large-scale GPS trajectories, that are labeled according to the proposed strategy in Section 3.3, is exploited for model evaluation. Before using the GPS data for training purposes, the following pre-processing steps are applied to each GPS trajectory so as to prepare them for generating the proposed GPS representation, introduced in Section 3.4, and remove erroneous GPS points caused by error sources in the GPS technology (e.g., satellite or receiver clocks).

- GPS points whose speed and/or acceleration exceeds a realistic value for vehicles moving in the US road networks are identified and discarded. The maximum speed and acceleration are set to 54 m/s and 10 m/s^2, respectively.

- After removing the unrealistic GPS points, the GPS trajectories containing less than 4 points are disregarded. Similarly, GPS trajectories shorter than 600 meters or 10 min were disregarded as well.
- The maximum length (i.e., the number of GPS legs) is set to 70, which is equivalent to the 80 percentile of the number of GPS legs in all GPS trajectories.
- After converting raw GPS trajectories into the proposed GPS representation, except for the road type binary features, all other types of features are standardized by subtracting the mean value and dividing by the standard deviation.

The number of labeled GPS trajectories per each vehicle class after applying the above-mentioned processing steps has been shown in the last column of Table 3.2. The processed GPS data are used for building our proposed model and baselines.

After pre-processing GPS trajectories, several variants of the original labeled dataset are created by re-grouping vehicle classes so as to examine and validate the proposed model. The vehicle-class definition is the only difference among the following datasets.

- **2&3** This dataset contains GPS trajectories only from FHWA class 2 and class 3. The proposed model needs to discriminate passenger cars (e.g., sedans, coupes, and station wagons) from all other two-Axle, four-tire single unit vehicles (e.g., campers, motor homes, and ambulances). As mentioned, the vehicles with class 2 and 3 of the FHWA system are categorized into one group (i.e., vehicles with gross weight less than 14,000 lbs) according to the GPS provider classification system. Hence, this type of problem is particularly useful for subdividing GPS trajectories with the gross weight under 14,000 lbs into the class 2 and class 3 of the FHWA system.
- **light&heavy** This dataset categorizes all GPS trajectories into two classes: (1) light-duty, which contains vehicles from class 2 and class 3, and (2) heavy-duty, which contains vehicles from all the other classes. This definition was first introduced in [80] and also used in [74].

- **light&medium&heavy** This dataset categorizes all GPS trajectories into three classes: (1) light-duty, which contains vehicles from FHWA class 2, (2) medium-duty, which contains vehicles from FHWA classes 3-6, (3) heavy-duty, which contains vehicles from FHWA class 7 and above. This classification is based on Gross Vehicle Weight Rating (GVWR) system, reported in the US Department of Energy Portal. Note that the corresponding FHWA vehicle classes in each category of GVWR is slightly different from weight category system defined by the GPS provider.

3.5.2 Baselines

Two types of baseline methods are considered for comparison: 1) classical-supervised methods on the top of hand-crafted features, 2) deep-learning models on the top of GPS representation proposed in Section 3.4.

3.5.2.1 Classical-supervised baselines In the first group, widely used supervised algorithms in the literature of GPS-based vehicle classification and travel mode detection are deployed including K-Nearest Neighborhood (KNN), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), as a representative of ensemble algorithms, and Multilayer Perceptron (MLP), as a regular fully-connected neural networks. Since the proposed GPS representation is not an appropriate input for these algorithms, a set of robust and efficient hand-designed features are extracted from each GPS trajectory. These features, which are introduced in the seminal study by [97] include the trajectory's length, mean speed, expectation of speed, variance of speed, top three speeds, top three accelerations, heading change rate, stop rate, and speed change rate. These are the most robust and acceptable hand-designed features in this domain that have been used by various studies.

3.5.2.2 Deep-learning baselines With the respect to the second group, several deep-learning models are developed for comparison. Major training blocks for building deep-learning baselines are CNN, RNN, and attention mechanism.

Recurrent Neural Network. An RNN is a chain-like neural network architecture that allows information to be passed from one step to another step. Thus,

they constitute a series of repeating modules of neural networks, where each module consists of a hidden state that operates on sequential data like GPS trajectory representation $\mathbf{X}$ [13]. For our proposed GPS representation, as shown in Figure 3.6, each module looks at the current input x_i, which is the feature vector corresponding to the i-th GPS leg in the GPS trajectory, and the previous hidden state to update the hidden state at the current module through a set of non-linear functions. Long Short Term Memory network (LSTM) is the most widely used module in RNN layers that utilizes several non-linear functions to update the hidden state in each RNN module. The hidden state in the last layer is often used as the output of an RNN layer. Further details on the RNN and LSTM models can be found in [13, 38].

Attention mechanism. Attention mechanism, first introduced by [5] is a new trend in deep-learning models that have received much interest in recent years. Attention helps a neural network to learn the importance of various regions of an input and focus more on certain input parts for performing the task-at-hand. It seeks to emulate the human's visual attention mechanism, which often pays more attention to only a small amount of information presented in visual scenes while doing their routine chores. Although a number of studies have assessed the efficacy of the attention mechanism in several applications including neural machine translation [5], text classification [21], text summarization [68], and caption generation [92], no study has examined the attention mechanism in the GPS trajectory mining tasks.

In particular for our application, the ultimate goal in the attention layer is to compute an importance weight for each abstract leg. Let $u \in \mathbb{R}^K$ be the attention vector, where K is the number of feature maps in the last CNN/RNN layer. Let $\mathbf{X}_{last}$ be the output of the last convolutional layer, in which $x_i \in \mathbb{R}^K$ is the feature vector corresponding to the i-th abstract GPS leg in $\mathbf{X}_{last}$. The importance of the abstract leg x_i is measured by computing its similarity with the attention vector u, and then get a normalized importance weight α_i through a softmax function. Elements in the attention vector u are trainable weights that are jointly learned with other network weights during the training process. Finally, the weighted representation of abstract legs are summed up to generate the attention vector representation, which is fed into a softmax layer for performing the classification

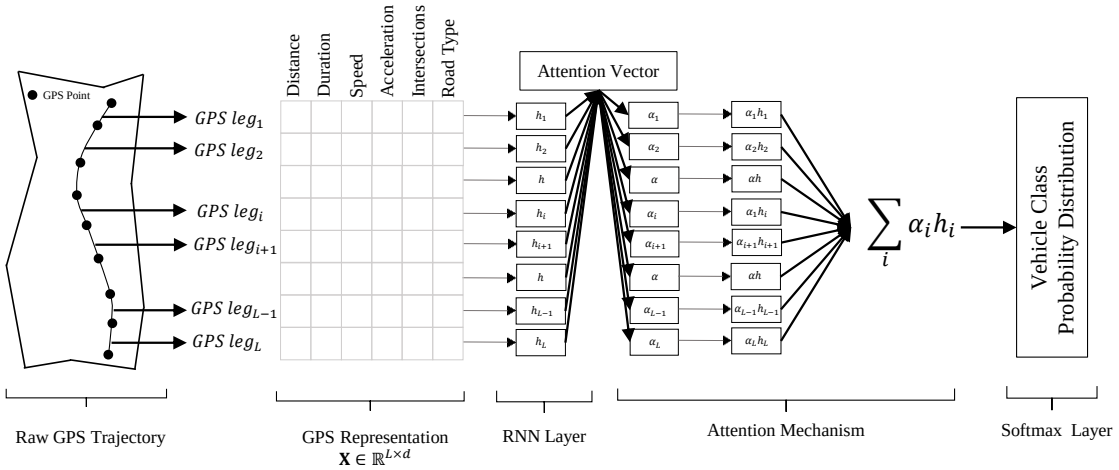


Figure 3.6: The structure of an RNN layer with attention mechanism. Analogously, the attention mechanism can be applied to the convolutional layers.

task. Figure 3.6 depicts how the attention mechanism works on the top of an RNN layer. Further details on the attention mechanism are available in the seminal studies by [5, 93].

Using CNN, RNN, and attention mechanism, the following deep-learning baselines are used, where all trained on the top of our proposed GPS representation:

- **RNN**: A single RNN layer with LSTM unit is used. The hidden state in the last LSTM unit is directly fed into a softmax layer for the classification task.
- **RNN-Attention**: The model structure is the same as the **RNN** baseline while an attention mechanism is used between the RNN and softmax layers, as shown in Figure 3.6.
- **Parallel-CNN**: Considering a convolutional layer followed by a max-pooling layer as one convolutional unit, several units are operated on the same GPS representation on a parallel way. The abstract feature-vector generated by each unit is then concatenated to create the final feature vector, which is fed into the softmax layer for the classification task. Note that convolutional and max-pooling operations are similar to our proposed model in Section 4.4. The primary difference between this baseline and our proposed architecture is that convolutional layers in our model are stacked on the top of each other and in turn operated sequentially rather than in a parallel way.

3.5.2.3 Performance Evaluation As mentioned, the vehicle classification represents an imbalanced classification problem. In this situation, a predictive model could simply achieve high accuracy by predicting all unseen records as the dominant class and considering samples in the minority class as noises. A robust and high-quality model needs to have a good prediction capability for both majority and minority classes. For imbalanced classification problems, recall is much more reliable performance metric that implicitly indicates the model accuracy for retrieving true positives corresponding to each class. Since we are seeking a model that is able to retrieve a majority of true positives for all vehicle classes, average recall is selected as the main performance measure. Average recall is computed based on one specific probability threshold (e.g., 0.5). However, the threshold value for classification might be subjected to change depending on application objectives. Area Under the Receiver Operating Characteristic curve (AUROC) is another reliable metric that, like recall, measures the model accuracy for each class. For a binary-class problem, Receiver operating characteristic Curve is created by plotting true positive rate (i.e., recall for positive class) versus false positive rate (i.e., 1− recall for negative class), for various threshold settings. The following performance metrics are used for models evaluation:

- *AVE-Recall.* AVE-Recall is the average recall values of all vehicle classes in a dataset, which is defined as below:

$$AVE - Recall = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FN_c}$$

  where TP_c and FN_c are the number of true positives and false negatives in the test set related to the class c. C is the total number of vehicle class in a dataset.

- *AUROC.* First, AUROC is computed for each class, and then the average AUROC for all classes is reported. For computing AUROC for each class in a multi-class problem, the subject class is considered as positive while the remaining assigned as negative.

- *Accuracy.* Accuracy is computed as the fraction of GPS trajectories in the test set that are correctly classified. While accuracy is not considered as the

main metric in this study on account of imbalanced distribution issue, we report this metric because it is the most common classification metric.

In all our 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-fold are reported. In other words, we create 5 train/test splits from the whole dataset. In each split, 4-fold is used for training while holding out the remaining one-fold as the test set. Before obtaining the average results on the 5 test folds, models' hyperparameters are tuned. To this end, one-fold out of the 4-fold training is used as the validation set for each split. The hyperparameter combination that on average achieves the highest performance on the validation sets of all splits is used for the final training and testing on 5 train/test splits. In this case, the test fold in each split plays no role in tuning hyperparameters. Next, using the obtained hyperparameters, models are trained on the 4-fold and tested on 1-fold for each train/test split and the average results are reported. By doing this, every GPS trajectory in the whole dataset is used in the test set at least and at most once. Only for implementing the early-stop method in deep-learning models, 10% of the training data in each split is randomly selected as the validation set for the early-stopping procedure using the stratified random selection.

All data processing and models are implemented within Python programming language. The deep-learning architectures are learned in TensorFlow with the GPU support. Classic machine-learning algorithms are implemented using the scikit-learn library. The source codes related to all data processing and models utilized in this study are available at: https://github.com/sinadabiri/CATT-DL-Vehicle-Classifcation-GPS.

3.5.3 Hyperparameter Settings

The number of convolutional sets, denoted as D, the filter size n, and the number of filters K in each layer are the primary hyperparameters in our proposed architecture. The grid search is a common approach for tuning hyperparameters that exhaustively considers all parameter combinations, yet it may not be an efficient way for deep-learning models with many hyperparameters. Instead, a manual

search is conducted by designing a variety of model configurations and selecting the best parameter combination. We tune the hyperparameters based on the AVE-Recall metric by examining the model performance on only the 2&3 dataset. Table 3.3 shows the average of AVE-Recall along with the standard deviation over 5 folds for several CNN-VC configurations. A wide range of configurations has been designed by varying the number of convolutional sets, filter size, and number of filters in the range of $D \in [1, 2, 3]$, $n \in [1, 2, 3, 4]$, and $K \in [25, 50, 75, 100, 125]$, respectively. As can be seen from Table 3.3, setting the filter size and the number of filters in each convolutional layer to 2 and 100, respectively, achieves the optimum performance. Furthermore, increasing the number of convolutional sets, as used in configurations #9-10, does not boost the performance. Thus, a network with one convolutional set is selected to avoid the model complexity and reduce the training time. Even using multiple sets of convolutional layers with different filter sizes and number of filters (i.e., configuration #11) does not ameliorate the model performance. Finally, comparison between configurations #4 and #12 demonstrates that using a max-pooling operation on the top of the last convolutional layer significantly improves the prediction quality by increasing the AVE-Recall by more than 4%.

Table 3.3: AVE-Recall for several CNN-VC configurations. The convolutional layer hyperparameters are denoted as conv(filter size)-(number of filters). Each conv set consists of two stacked convolutional layers with the same hyperparameters. NA: Not Available

#Config	Conv-set 1	Conv-set 2	Conv-set 3	Max-Pool	AVE-Recall
1	Conv2-25	NA	NA	Available	0.793(±0.003)
2	Conv2-50	NA	NA	Available	0.805(±0.007)
3	Conv2-75	NA	NA	Available	0.807(±0.004)
4	**Conv2-100**	**NA**	**NA**	**Available**	**0.819(±0.005)**
5	Conv2-125	NA	NA	Available	0.811(±0.007)
6	Conv1-100	NA	NA	Available	0.809(±0.008)
7	Conv3-100	NA	NA	Available	0.798(±0.007)
8	Conv4-100	NA	NA	Available	0.777(±0.012)
9	Conv2-100	Conv2-100	NA	Available	0.789(±0.010)
10	Conv2-100	Conv2-100	Conv2-100	Available	0.784(±0.016)
11	Conv2-50	Conv3-75	Conv4-100	Available	0.791(±0.004)
12	Conv2-100	NA	NA	NA	0.776(±0.007)

Table 3.4: Comparison of AVE-Recall values on created datasets using classical-supervised algorithms, deep-learning baselines, and our proposed CNN-VC model.

Model \ Dataset	**2&3**	**light&heavy**	**light&mid&heavy**
KNN	0.566 (±0.005)	0.568 (±0.001)	0.392 (±0.001)
SVM	0.623 (±0.024)	0.582 (±0.010)	0.421 (±0.014)
DT	0.576 (±0.004)	0.616 (±0.001)	0.438 (±0.001)
RF	0.555 (±0.004)	0.637 (±0.002)	0.467 (±0.002)
MLP	0.649 (±0.012)	0.624 (±0.010)	0.437 (±0.006)
RNN	0.766 (±0.048)	0.732 (±0.002)	0.615 (±0.011)
RNN+Attention	0.776 (±0.014)	0.732 (±0.002)	0.619 (±0.002)
CNN-Parallel	0.752 (±0.008)	0.701 (±0.004)	0.573 (±0.003)
CNN-VC+Attention	0.812 (±0.011)	0.735 (±0.003)	0.614 (±0.002)
CNN-VC (ours)	**0.819 (±0.005)**	**0.741 (±0.005)**	**0.631 (±0.004)**

3.5.4 Comparison results

Tables 3.4, 3.5, and 4.2 summarize the average results of the stratified 5-fold cross-validation along with the standard deviation for our CNN-VC model and baselines in terms of AVE-Recall, AUROC, and accuracy, respectively. In addition, the impact of the attention mechanism in our CNN-VC model is investigated by adding the attention layer, as shown in Figure 3.6, on the top of the convolutional layers. Since the attention layer summarizes the high-level GPS representation into a feature vector for passing into the softmax layer, the max-pooling operation is taken out from the network. Like our CNN-VC model, the hyperparameters associated with every baseline is first tuned on the 2&3 datasets. Then, using the optimum combination of hyperparameters, every model is trained and tested on the three datasets: 2&3, light&heavy, and light&medium&heavy.

Considering AVE-Recall and AUROC as the most reliable metrics, Tables 3.4 and 3.5 clearly show the superiority of our CNN-VC model in comparison with both classical-supervised and deep-learning models for all three datasets. With respect to AVE-Recall, the CNN-VC model achieves on average 19% and 4% better performance compared to the classical-supervised and deep-learning models, respectively. Analogously, the CNN-VC surpasses the baselines by obtaining on average 22% and 3% higher AUROC in comparison with the classical-supervised and deep-learning models, respectively.

Table 3.5: Comparison of AUROC values on created datasets using classical-supervised algorithms, deep-learning baselines, and our proposed CNN-VC model.

Model \ Dataset	**2&3**	**light&heavy**	**light&mid&heavy**
KNN	0.585 (±0.005)	0.593 (±0.001)	0.557 (±0.001)
SVM	0.692 (±0.022)	0.649(±0.009)	0.603 (±0.010)
DT	0.576 (±0.004)	0.619 (±0.001)	0.575 (±0.001)
RF	0.737 (±0.004)	0.687 (±0.002)	0.634 (±0.001)
MLP	0.685 (±0.012)	0.655 (±0.011)	0.591 (±0.007)
RNN	0.873 (±0.044)	0.830 (±0.004)	0.797 (±0.007)
RNN+Attention	0.884 (±0.008)	0.831 (±0.004)	0.801 (±0.001)
CNN-Parallel	0.845 (±0.006)	0.789 (±0.004)	0.758 (±0.002)
CNN-VC+Attention	0.909 (±0.003)	0.832 (±0.001)	0.798 (±0.002)
CNN-VC	**0.910 (±0.003)**	**0.840 (±0.002)**	**0.810 (±0.002)**

What is striking about Tables 3.4 and 3.5 is the significant superiority of deep-learning models compared to classical machine-learning techniques. One of the key differences between these two architectures is the structure of their input volume, the proposed GPS representation versus a set of hand-crafted features. Since the ultimate performance of a machine-learning algorithm is highly contingent on the quality of its input representation, the superiority of deep-learning models strongly demonstrates the effectiveness of our proposed GPS representation. Another interesting finding is that using the attention mechanism on the top of RNN and CNN blocks could not improve the model performance. This is a compelling reason to utilize the pooling operation instead of the attention mechanism in our proposed network. The comparison between RNN and CNN based models reveals that although the RNN architecture generates competitive results, our CNN-VC outperforms by improving the AVE-Recall and AUROC on average 3% and 2%, respectively. Finally, the clear superiority of CNN-VC compared to CNN-Parallel demonstrates that stacking convolutional layers in sequence ameliorates the overall performance of the model, against deploying convolutional layers in parallel.

Although the accuracy is not a reliable metric for the imbalanced problems, we do report the accuracy results in Table 4.2 to evaluate the overall performance of our model in terms of this widely used metric. As can be seen, except for RF, our CNN-VC model works better than all baselines for three datasets. While the

Table 3.6: Comparison of Accuracy values on created datasets using classical supervised algorithms, deep-learning baselines, and our proposed CNN-VC model.

Model \ Dataset	**2&3**	**light&heavy**	**light&mid&heavy**
KNN	0.672 (±0.002)	0.611 (±0.001)	0.395 (±0.001)
SVM	0.658 (±0.019)	0.534 (±0.027)	0.337 (±0.055)
DT	0.814 (±0.002)	0.737 (±0.001)	0.511 (±0.002)
RF	**0.869 (±0.002)**	**0.790 (±0.001)**	0.589 (±0.002)
MLP	0.493 (±0.053)	0.764 (±0.058)	0.602 (±0.023)
RNN	0.763 (±0.165)	0.751 (±0.022)	0.607 (±0.048)
RNN+Attention	0.853 (±0.022)	0.749 (±0.021)	0.587 (±0.025)
CNN-Parallel	0.720 (±0.041)	0.670 (±0.029)	0.562 (±0.036)
CNN-VC+Attention	0.811 (±0.043)	0.771 (±0.011)	0.596 (±0.019)
CNN-VC	0.856 (±0.016)	0.771 (±0.025)	**0.610 (±0.004)**

model is forced to have close prediction quality for all vehicle classes, it achieves high and reasonable accuracy as well. The high accuracy value in contrast to low AVE-Recall and AUROC values for some baselines (e.g., RF and DT) is another proof that using only accuracy measure for imbalanced problems is not sufficient for model evaluation.

3.5.5 Effect of GPS Representation

As noted, the comparison results have already demonstrated the effectiveness of our proposed GPS representation. In addition, the effect of motion features and roadway features are investigated by training our model on the top of different variants of GPS representation: (1) only motion features, (2) only roadway features, and (3) combination of motion and roadway features. Table 3.7 shows the average results for variants of GPS representation on the 2&3 dataset. From Table 3.7, it is apparent that the impact of motion features in improving the model quality is more significant. However, fusing motion features with roadway characteristics improves the prediction quality by more than 2% in terms of AVE-Recall, as the main metric value. Also, it should be noted that the GPS representation structure is very adaptable to be combined with information from other sources (e.g., environmental data) upon availability.

Table 3.7: Performance comparison of the CNN-VC model with variants of GPS representation on the 2&3 dataset

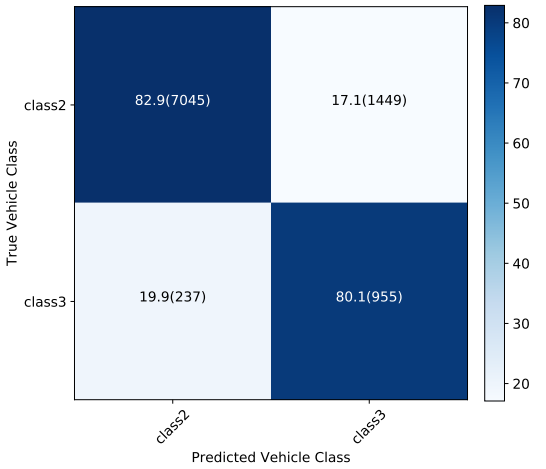
GPS Representation	AVE-Recall	AUROC	Accuracy
Only motion features	0.794 (±0.011)	0.896 (±0.004)	0.859 (±0.029)
Only roadway features	0.690 (±0.005)	0.771 (±0.005)	0.667 (±0.023)
Motion and roadway features	**0.819 (±0.005)**	**0.910 (±0.003)**	**0.856 (±0.016)**

3.5.6 Model performance interpretation

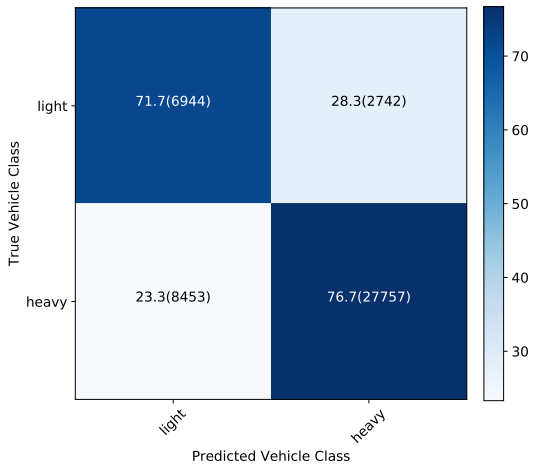
An interesting analysis is to investigate the model prediction quality for every vehicle class. Confusion matrix, as shown in Figure 3.7 for all types of datasets, allows us to visualize our CNN-VC quality at retrieving true positives for each vehicle class. Values in diagonal elements of Figure 3.7 represent the percentage of GPS trajectories per each class in the test set that the CNN-VC model has correctly predicted (i.e., the recall value for each vehicle class). On the other hand, the off-diagonal values in each row shows the misclassification rate. The values in the parenthesis represent the number of true GPS trajectories that were assigned to various classes.

As shown in Figure 3.7a, the CNN-VC achieves almost the same performance for correctly predicting both the majority vehicle class (i.e., class 2) and the minority vehicle class (i.e., class 3), with recall values around 81%. It should be noted that the FHWA class 2 and class 3 include vehicles with similar operating functions and in turn similar moving patterns. For example, pickups and vans, which are categorized as the FHWA class 3, may be utilized as passenger cars, which is the main function of vehicles in the FHWA class 2. Accordingly, a high-quality performance for discriminating between the FHWA class 2 and 3 is not a trivial task, yet it has been attained by the CNN-VC model. Since the raw GPS data collected by the GPS provider categorizes these two classes in the same weight group (i.e., less than 14,000 lbs), a practical benefit of the classifier on the 2&3 dataset is to distinguish the GPS trajectories of class 2 from class 3 across the entire dataset including 20 million GPS traces.

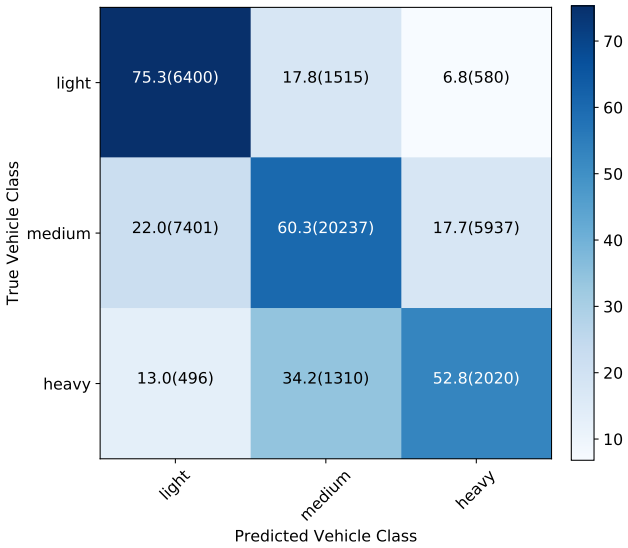
In a similar fashion, it can be observed from Figure 3.7b that the proposed model



(a) Confusion matrix for 2&3 dataset



(b) Confusion matrix for light&heavy dataset



(c) Confusion matrix for light&medium&heavy dataset

Figure 3.7: Confusion matrices of the CNN-VC model for three datasets.

has also a reasonable performance when vehicles are categorized into light-duty versus heavy-duty. The model's capability for detecting both the non-dominant class (i.e., light class in this dataset) and the dominant-class (i.e., heavy class in this dataset) are roughly similar, with the recall values around 74%. It should be emphasized that even the FHWA 13-category rule sets have some difficulty in distinguishing between vehicle categories when the information on the number, weight, and spacing of axles are available by means of traffic flow sensors [37]. For example, a pickup truck with conventional two-tire real axle (FHWA class 3)

cannot be distinguished from the similar truck yet with dual tries on each side of its rear axle (FHWA class 5) when they are empty. This problem is exacerbated when the information related to the vehicle is limited to only a sequence of GPS records, rather than vehicle shape and weight characteristics. Considering such issues, an accuracy around 74% for both light-duty and heavy-duty vehicle classes can be considered as an acceptable performance while all information is limited to the vehicle mobility pattern obtained from GPS records.

As expected and shown in Figure 3.7c, the performance quality is worsened by increasing the number of vehicles in the light&medium&heavy dataset. The CNN-VC achieves a good performance in distinguishing light (i.e., FHWA class 2 and 3) and medium (i.e., FHWA class 3-6) vehicles. However, the model encounters difficulty for discriminating the heavy vehicles from medium ones, where 34% of heavy-duty vehicles are misclassified as medium-duty. Although a part of this low-quality performance stems from the quality of the original GPS data and the overall prediction ability of a machine-learning algorithm, distinguishing between medium and heavy vehicles is not a trivial task even when information on vehicles' axles is available. For instance, a light truck pulling utility trailer, which is classified as the FHWA class 3 and medium-duty class, may have similar axle configuration to a truck pulling a heavy single-axle trailer, which is classified as the FHWA class 8 and heavy-duty class [37]. Unless accessing to the axle weight information, these two trucks cannot be differentiated from each other even using fixed-point traffic flow sensors. Furthermore, trucks size and weight configurations might vary among different States and manufacturers, depending on State laws and companies profit strategies although they are designed for exactly the same purposes. Considering the boundary vehicles types (e.g., the FHWA class 6 as the medium-duty class against the FHWA vehicle class 7 as the heavy-duty class), that have almost the same operation capabilities, is sufficient to intuitively imagine the difficulty in distinguishing these boundary classes using only GPS information.

In summary, the GPS sensor is a reliable alternative to overcome the shortcomings of fixed-point sensors by reducing the installation and maintenance costs, removing the spatial coverage limitation, and opening the opportunity for online monitoring the traffic network by inferring the vehicle-class distribution. However, it is obvious that GPS trajectories can only represent the mobility patterns of moving

vehicles around the road network. Lack of information on the axle and weight configuration, which cannot be extracted from GPS data, makes the GPS-based vehicle classification harder than alternative methods, in particular for fine-grained classification scheme with several vehicle categories. While the results in this study show that building a coarse-grained vehicle-classification scheme using GPS data is achievable, creating a robust model for the fine-grained classification scheme requires additional information. A potential solution for improving the model performance while preserving the advantages of the GPS data (e.g., wide-area spatial coverage) is to fuse other information such weather conditions, that manifests the road network with more details, with the GPS data. As described in Section 4.4, our proposed GPS representation has the flexibility for including more information about the vehicle and road network along the GPS path.

3.5.7 Practical Applications

The proposed method can be readily used to label all 20 million trajectories that Maryland State Highway Administration is using to support its analysis and decision making. Such enriching of the trajectory data would enable more comprehensive studies that require information about vehicle classes. Examples include, but are not limited to:

- Derivation of origin-destination tables for individual FHWA vehicle classes, which is especially useful for planning efforts where transportation analysts need to distinguish between travel patterns of passenger cars used mainly for commuting and commercial vehicles;

- Examining whether trajectories associated with heavy trucks are observed in downtown areas or along the routes with weight restrictions, which would indicate the need for additional enforcement in those areas in order to establish the desired level of safety;

- Analyzing whether trajectories attributed to large trucks are deviating from locations with weigh-in-motion systems, which would suggest the need for deploying mobile patrols in these regions to enforce weight limits;

- Estimation of FHWA-class-specific volumes along different road links in a transportation network, which would allow transportation analysts to more accurately measure performance as it would enable them to distinguish between vehicle-hour and truck-hour delays;
- More accurate estimation of emissions based on trajectory data, where individual tracks can now be attributed to different vehicle classes (e.g., passenger cars vs. trucks), which obviously makes a big difference in estimating transportation-related emissions.

3.6 Conclusion

Vehicle classification, as an essential step in a variety of ITS applications, has been addressed for decades using fixed-point traffic flow sensors. However, such conventional senors suffer from major shortcomings including high maintenance cost and low spatial coverage. To address these shortcomings, we aimed to leverage the spatiotemporal information of vehicles' trips retrieved through on board GPS-enabled devices, for classifying vehicles into various categories. First, we designed an efficient programmatic approach to label a large scale GPS data with the aid of an auxiliary resource, (i.e., VWS vehicle records). Using the labeled GPS data, we proposed a deep convolutional neural network (CNN-VC) for identifying the vehicle class from its GPS trajectory. Since a GPS trajectory contains a sequence of GPS coordinates without any meaningful information, a novel representation was designed to convert the GPS trajectory into a matrix representation. Each row in the new representation corresponds to motion-related and roadway-related features for the segment between two consecutive GPS points. Afterward, a stack of convolutional layers, followed by a max-pooling opertation, were applied on the top of the proposed GPS representation so as to compute the abstract representation of the GPS trajectory for the classification task using the softmax operation. Our extensive experiments clearly demonstrated the superiority of the CNN-VC model in comparison with several state-of-the-art classical-supervised and deep-learning techniques.

Chapter 4

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

Abstract

Identification of travelers' transportation modes is a fundamental step for various problems that arise in the domain of transportation such as travel demand analysis, transport planning, and traffic management. In this paper, we aim to identify travelers' transportation modes purely based on their GPS trajectories. First, a segmentation process is developed to partition a user's trip into GPS segments with only one transportation mode. 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, we propose a deep **SE**mi-Supervised **C**onvolutional **A**utoencoder (**SECA**) architecture that can not only automatically extract relevant features from GPS segments but also exploit useful information in unlabeled data. The SECA integrates a convolutional-deconvolutional autoencoder and a convolutional neural network into a unified framework to concurrently perform supervised and unsupervised learning. The two components are simultaneously trained using both labeled and unlabeled GPS segments, which have already been converted into an efficient representation for the convolutional operation. An optimum schedule for varying the balancing parameters between reconstruction and classification errors are also implemented. The performance of the proposed SECA model, trip segmentation,

the method for converting a raw trajectory into a new representation, the hyper-parameter schedule, and the model configuration are evaluated by comparing to several baselines and alternatives for various amounts of labeled and unlabeled data. Our experimental results demonstrate the superiority of the proposed model over the state-of-the-art semi-supervised and supervised methods with respect to metrics such as accuracy and F-measure.

Keywords: Deep learning; semi-supervised learning; convolutional neural network; convolutional autoencoder; GPS trajectory data; trip segmentation, transportation mode identification.

4.1 Introduction

The mode of transportation for traveling between two points of a transportation network is an important aspect of users' mobility behavior. Identifying users' transportation modes is a key step towards many transportation related problems including (but not limited to) 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 [89]. 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 spatio-temporal information regarding human activities, provokes several transport-domain applications such as incident detection, mobility pattern extraction, and transport mode inference [18]. In this study, 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 [91, 97, 98]. 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 [89].

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 under a congested traffic 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. Recently, researchers have shown an increased interest in leveraging deep learning algorithms for addressing challenging transportation-related problems [16, 26, 88].

Figure 4.1 depicts the overview of our framework for identifying the transportation mode(s) of a GPS trajectory related to a user's trip. Since travelers might commute with more than one transportation mode for making a single trip, the first step is to partition the GPS trajectory of a trip into segments, in which every GPS segment contains only one transportation mode. Next, features of each GPS segment is automatically extracted using a deep learning framework based on Convolutional Neural Network (CNN). Accordingly, one of our main challenges is to convert the raw GPS segment into an adaptable layout for CNN schemes. First the basic motion characteristics of every GPS point in a segment including relative distance, speed, acceleration, and jerk are computed [17]. This results in generating a sequence of numerical features for every type of motion characteristic. Next, the computed motion sequences are concatenated to create a 4-channel tensor for every GPS segment, where every sequence is equivalent to a channel in

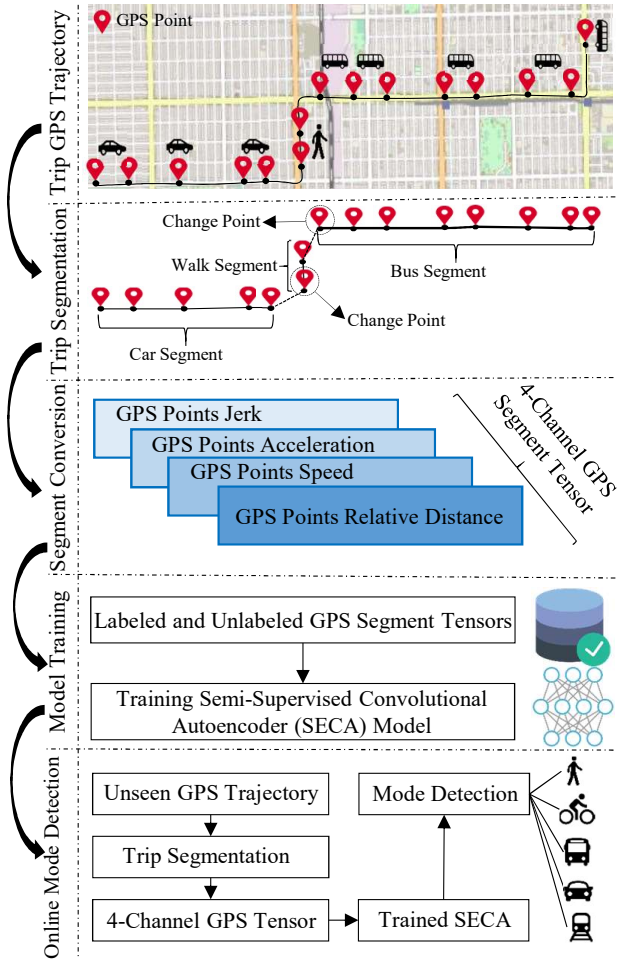


Figure 4.1: Overview of our framework for detecting transportation mode of GPS trajectories. A raw GPS trajectory of a user's trip is first partitioned into a set of GPS segments with only one transportation mode. Next, each GPS segment is converted to a 4-channel tensor. The created labeled and unlabeled tensors are then used for training our proposed SECA model. The trained SECA model is finally used for detecting the transportation mode(s) of an unseen GPS trip.

an RGB image. Such a new representation 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 representation 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. Finally, the transportation mode of a GPS segment is inferred by training a CNN-based deep learning architecture on the converted GPS segments.

Moreover, almost all the current research work on travel mode identification has built their models using only labeled trajectories. Nonetheless, a significant portion of trajectories in GPS data might not be 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 allows us to capture more properties of the data distribution, which can potentially further improve the decision boundaries and provide a better generalization on unseen records [47]. 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). A deep **SE**mi-Supervised **C**onvolutional **A**utoencoder (**SECA**) architecture, that integrates 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:

- **Developing a two-step segmentation process.** Due to the inherent need of CNN-based models for having a fixed-size input, the GPS trajectory of a trip is first uniformly partitioned into the GPS segments with a fixed size. Afterward, for the first time in this domain, a discrete optimization algorithm is deployed to detect the points where the transportation mode changes. The output of this step is a pool of GPS segments with only one transportation mode.

- **Designing an efficient representation for raw GPS trajectories.** A new procedure is developed for converting a raw GPS segment, which is a sequence of GPS points, to an efficient and appropriate representation for using in deep learning architectures. The proposed representation contains the information of *all* GPS points in the GPS Segment and allows the very deep architecture and training algorithm to extract discriminating and high-level features for the task at-hand.

- **Developing a novel deep semi-supervised convolutional autoencoder (SECA) architecture.** A deep semi-supervised model is proposed

to leverage both *unlabeled* and labeled GPS trajectories for predicting transportation modes. The model contains Conv-AE and CNN classifier for unsupervised and supervised learning, respectively.

- **Building an effective schedule for tuning balancing parameters.** Since the main objective is to simultaneously training the unsupervised and supervised components of the SECA model, a novel and efficient schedule is proposed for varying the balancing parameters that combine reconstruction and classification losses.

- **Conducting an extensive set of experiments for performance evaluation and comparison.** The results reveal that our SECA model outperforms several supervised and semi-supervised state-of-the-art baseline methods for various amounts of labeled GPS segments. Furthermore, the performance results demonstrate the superiority of the proposed trip segmentation process, the designed representation for GPS segments, the schedule for varying hyperparameters, and the model structure by comparing with several alternatives.

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 mode detection framework are explained in Section 4. Our experimental results are reported in Section 5. Finally, the paper is concluded in Section 6.

4.2 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 [26, 51]. In this section, we review the studies that have utilized the GPS data for designing a mode detection model since this is the primary focus of this paper. A comprehensive and systematic review of existing techniques for travel mode recognition based on GPS data is available in [89]. 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 detection models, we will also briefly discuss various semi-supervised deep learning architectures that have been studied in the literature for different applications.

4.2.1 GPS-Based Mode Detection Models

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

In two seminal studies by Zheng *et al.* [97, 98], a supervised framework based on hand-crafted features was proposed. Using the commonsense knowledge of the real world, a trip is first partitioned into segments by detecting the walk segments. Then, a set of manual yet robust features were identified for every segments 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 contains descriptive statistics of velocity and acceleration of all GPS points while the robust group 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.* [91] 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. Mäenpää *et al.* [57] 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. Nevertheless, research on semi-supervised mode inference is really scarce, Rezaie *et al.* [66] 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 [17, 24, 85]. After converting a raw GPS trajectory into a matrix with the image format, 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 upon simultaneously using both labeled and unlabeled trajectories. Furthermore, no optimization technique has been exploited for trip segmentation.

4.2.2 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 [42, 64]. 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 entire network) are concurrently trained.

One technique for the pre-training phase is to sequentially train the network layers [7]. 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 [25]. In the next stage, the deep architecture is fine-tuned by performing a local search from a reasonable starting point. Another pre-training strategy is to train the network in its entirety, rather than greedy layer-wise training, and then fine-tune the model using weights obtained from the first phase as an initialization. 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 earlier, one of the main objectives in the pre-training step is to prepare a good initialization for the supervised training and avoid a poor general-

ization of the model. However, with advances in initialization and regularization schemes for deep learning architectures [30, 78], pre-training strategies are getting replaced with joint training strategies. The fundamental idea of 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 [47], convolutional-deconvolutional autoencoders [95], autoencoders based on a Ladder network [64], and recursive autoencoders [76] 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 step, the unlabeled data are annotated using the updated weights in the previous iteration [49]. A balancing parameter can be used in the hybrid loss function to trade off the supervised and unsupervised parts of the objective function [76, 95]. In Section 4.5, the effect of balancing parameter(s) in the ultimate performance of joint training strategies is examined.

In addition to be designed for a new application with a unique model configuration, our SECA model is trained using a new schedule for tuning balancing parameters. Unlike similar approaches in literature, the proposed schedule considers a separate balancing parameter for each of unsupervised and supervised components.

4.3 Preliminaries

This section introduces the preliminaries required to comprehend the proposed framework in Section 4.4. First, the trip segmentation and mode detection problems are described and then, we show how to compute the motion characteristics of each GPS point. The motion features of GPS points are then used for both creating an input layer in our SECA model and hand-crafted features in the standard machine learning algorithms.

4.3.1 Definitions and Problem Statements

Before describing the formal statements of the two problems, the notions of GPS trajectory and GPS segment are defined.

Definition 1 (GPS Trajectory). *A user's raw GPS trajectory T is defined 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.*

T is divided into *trips* if the time interval between two consecutive GPS points exceeds a pre-defined threshold (e.g., 20 minutes) [98]. 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, a trip is partitioned into multiple GPS segments when the transportation mode changes.

Definition 2 (Change Point). *A change point, denoted as CP, is defined as the place in a trip in which users change their transportation mode. A trip may contain zero, one, or multiple change points.*

Definition 3 (GPS Segment). *A GPS segment is a sub-division of a user's trip, which is traveled by only one transportation mode $y \in Y$, where Y is a set of transportation modes (such as bike and car). A GPS segment is denoted as $SE = [p_1, \cdots, p_M]$, where M is the number of GPS points that forms SE.*

Accordingly, the trip segmentation problem is defined as follows:

Problem 1: Trip Segmentation *The trip segmentation problem is defined as detecting the change points CP in the GPS trajectory of a user's single trip. GPS segments SEs with a unique transportation mode are the output this problem.*

The mode detection is a multi-class classification problem that seeks for predicting the correct transportation mode for a given 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 (for traditional machine learning algorithms) or a proper layout (for deep learning architectures), represented as $\mathbf{X}$, will need to be designed. The first step for generating $\mathbf{X}$ is to compute the motion characteristics of GPS points in each SE, which is described in the next section.

Problem 2: Mode Detection *Given the training data* $\{(\boldsymbol{X}_i, y_i)\}_{i=1}^{n}$ *for* n *samples of* SE_i*, the mode detection problem is defined as building the optimal classifier for estimating the transportation mode of a user's* SE *based on its features* $\boldsymbol{X}$*.*

The trained model is then deployed to estimate users' transport modes while traveling in transportation networks. For an unseen trip, a trip is first segmented into a set of SEs using the proposed trip segmentation technique. Then, the mode for each SE is estimated. Consecutive segments with the same detected modes are concatenated together.

4.3.2 Motion Characteristics of GPS Points

Several motion features can be computed for every GPS point based on their geographic coordinates and timestamps. The relative distance between two consecutive GPS points in a SE (e.g., p_1 and p_2), can be computed using the widely-used Vincenty's formula [84]. The Vincenty's formula is a common and accurate method for computing the geographical distance between two points on the surface of a spheroid. The time interval between two successive GPS points is another motion quantity that can be simply computed. Having the relative distance and time interval, other fundamental kinematic motions including speed, acceleration, and jerk are computed to provide more information about a user's motion. Speed is the rate of change in distance that shows how fast a user is traveling. Acceleration is the rate of change in speed that shows how fast a user is changing their speed. Jerk, the rate of change in acceleration, is a significant factor in safety issues such as critical driver maneuvers and passengers' balance in public transportation vehicles [4]. Jerk has been used in mode detection models for the first time in [17]. These motion features for a GPS point p_1 are calculated based on the following equations:

$$RD_{p_1} = Vincenty\big(p_1[lat, lon], p_2[lat, lon]\big) \tag{4.1}$$

$$\Delta t_{p_1} = p_2[t] - p_1[t] \tag{4.2}$$

$$S_{p_1} = \frac{RD_{p_1}}{\Delta t_{p_1}} \tag{4.3}$$

$$A_{p_1} = \frac{S_{p_2} - S_{p_1}}{\Delta t_{p_1}} \tag{4.4}$$

$$J_{p_1} = \frac{A_{p_2} - A_{p_1}}{\Delta t_{p_1}} \tag{4.5}$$

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. Analogously, the above-mentioned formulae are used to calculate the motion features of other GPS points in a SE.

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 [97]. Bearing rate is a motion attribute for quantifying the heading change among modes. As depicted in Fig. 4.2, bearing measures the angle between the line connecting two successive points and a reference (e.g., the magnetic or true north). The bearing rate, which can be used as another motion feature, is the absolute difference between the bearings of two consecutive points.

4.4 The Proposed Framework

In this section, first, a two-step process is proposed to divide the GPS trajectory of a user's trip into the segments with only one transportation mode. Next, an efficient representation for each GPS segment is designed. The overall architecture of our semi-supervised framework is also explained in detail. Finally, an effective strategy for training our network is proposed.

4.4.1 Two-Step Trip Segmentation

Our proposed method for partitioning a user's trip into segments with a unique transportation mode consists of two parts. Considering the basic CNN requirement

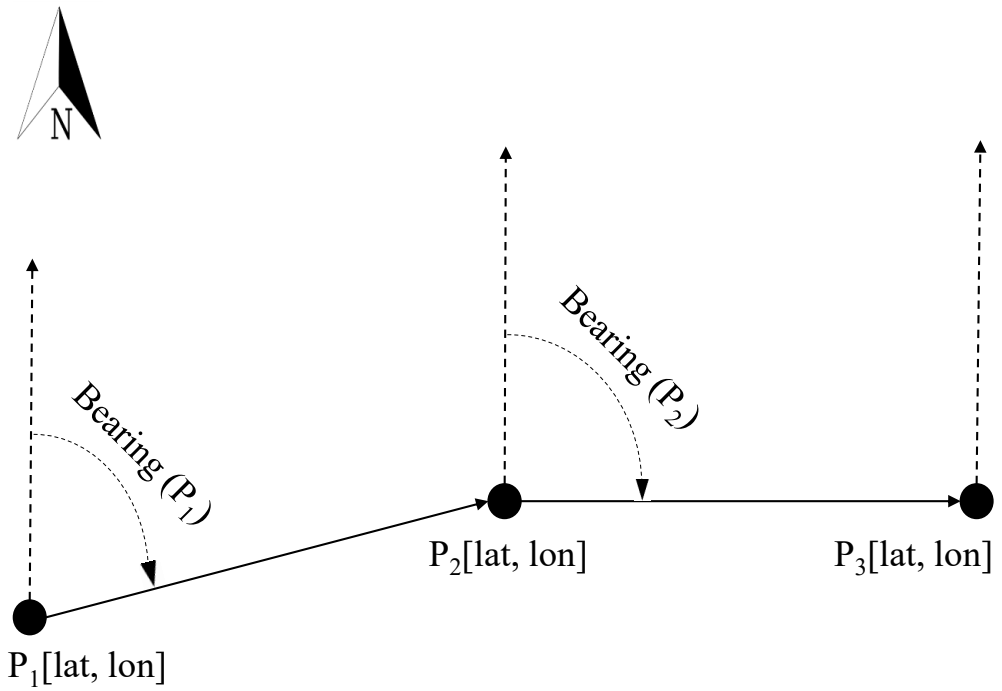


Figure 4.2: Bearing between two consecutive GPS points. *lat* and *lon* represents the latitude and longitude of a GPS point.

for having all input samples with a fixed size, all GPS trajectories need to be either truncated or padded to a fixed size for both offline learning and online inference at the end. Accordingly, we flip this requirement into the first step of our trip segmentation. Thus, a GPS trip is first uniformly partitioned into segments with a fixed number of GPS points, denoted as M. Our observation indicates that a majority of GPS segments contains one or two transportation modes after this uniform-size segmentation, which dramatically improves the performance of the overall segmentation process. In the online mode detection, consecutive segments with the same predicted transportation mode are merged together.

However, the uniform-size segmentation in the first step does not guarantee that every fixed-size GPS segment contains only one transportation, which calls for the second segmentation step. Assuming the fixed-size GPS segment $SE = [p_1, \cdots, p_M]$ as a signal, we aim to detect a number of change points, K, with their positions, $\mathbf{CP} = [CP_1, \cdots, CP_K]$, where the statistical properties of the signal change. Note that every change point belongs to $SE = [p_1, \cdots, p_{M_1}]$. To this end, SE is first converted into a multivariate time series $\{\mathbf{Y}_t\} \in \mathbb{R}^{(M \times d)}$, where

d is the number of motion features introduced in Section 4.3.2. Our observation indicates using only speed and acceleration features for creating $\{\mathbf{Y}_t\}$ results in a better performance. A common approach for detecting the change points from a time-series signal is to minimize the following objective function [44]:

$$C(\mathrm{CP}) = \sum_{i=1}^{K+1} [c(\mathbf{Y}_{CP_{i-1}+1:CP_i})] + \gamma f(K) \tag{4.6}$$

where $CP_0 = 0$ and $CP_{K+1} = M$. c is a cost function that measures the homogeneity in the sub-segments. We choose the mean-shift model, as one of the most studied cost function used in the change point detection literature [83]. For a GPS sub-segment $\{\mathbf{Y}_t\}_I$ on a time interval I between two consecutive change points, the mean-shifts is defined as follows:

$$c(\mathbf{Y}_I) = \sum_{t \in I} ||\mathbf{Y}_t - \bar{\mathbf{Y}}||_2^2 \tag{4.7}$$

where $\bar{\mathbf{Y}}$ is the mean of $\{\mathbf{Y}_t\}_{t \in I}$.

Since the number of change points K is unknown in our problem (i.e., a trip might be performed with various number of transportation modes), a penalty function $f(K)$ is used in Eq. 4.6 to constrain the number of change points. We choose a linear penalty function as γK. The penalty level γ makes a balance between the decrease in the cost function when more change points are allowed. Accordingly, solving the optimization problem in Eq. 4.6 is dependent on the choice of the penalty level γ, not a pre-defined number of change points.

A wide range of search algorithms have been proposed to solve the discrete change point detection optimization problem, formulated in Eq. 4.6 [83]. Our choice of the search algorithm is the Pruned Exact Linear Time (PELT) method, which has been recently proposed in [44]. PELT leverages the advantages of alternative search algorithms, which are exact solution and low computational complexity achieved through a combination of optimal partitioning and pruning. Under the assumption that change points are spread throughout the signal rather than confined to one portion, the PELT algorithm discards many points along the signal through its pruning step. This results in dramatically reducing the computational cost to on average $\mathcal{O}(n)$ while keeping the ability to detect the optimal segmentation. The

detailed algorithm can be found in [44].

If PELT finds a change point in a GPS SE, the SE is partitioned into two or more sub-segments. After converting each SE into a 4-channel tensor, as described in the following section, all short sub-segments are padded with zero values to have a fixed-size M before feeding into our SECA model.

4.4.2 New Representation for Raw GPS Segments

Since the core component of our proposed framework is a convolutional network, we will need to first convert GPS segments into a format that is not only compatible with CNN architectures but is also efficient in representing the fundamental motion characteristics of a moving object. As explained earlier, 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 placing the corresponding value for every GPS point of a SE in chronological order, where the feature value is computed using Eqs. (1)-(5). Such a sequence can be seen as a 1-d channel. Stacking these channels turns a raw GPS SE into a 4-channel tensor. In Section 4.5, we demonstrate the superiority of such a configuration compared to other possible feature combinations.

By padding the short segments with zero values, all GPS SEs are mapped into a 4-channel tensor with the shape of $(1 \times M \times 4)$. We select M as the median size of all GPS segments obtained based on the *true* change points, which results in the best final performance. Figure 4.3 illustrates the 4-channel arrangement 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 detection. 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.

4.4.3 Semi-Supervised Convolutional Autoencoder (SECA) Model

As can be seen in Fig. 4.4, our semi-supervised architecture combines two main components: (1) a CNN classifier, which takes in only the labeled trajectories,

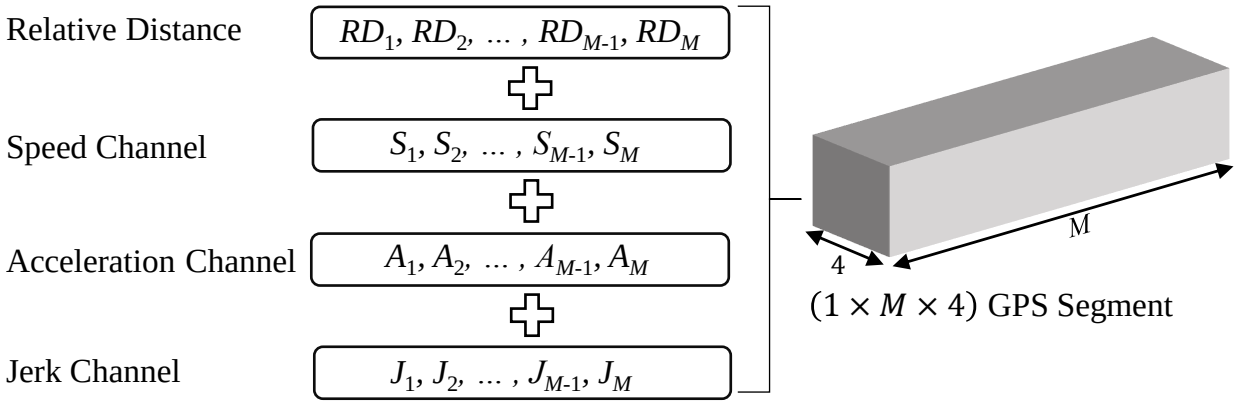


Figure 4.3: A 4-channel representation for a GPS segment with a shape of $(1 \times M \times 4)$

denoted as $\mathbf{X}_l \in \mathbb{R}^{(1\times M\times 4)}$, and (2) Convolutional-deconvolutional AutoEncoder (Conv-AE), which takes in both 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 $\mathbf{X}_u$ correspond to the 4-channel tensors of labeled and unlabeled GPS segments, respectively, which are created based on the procedure described in the previous section.

4.4.3.1 Convolutional-Deconvolutional AutoEncoder Autoendocer is an unsupervised learning technique that aims to learn an efficient latent representation by reconstructing the input at the output layer. Autoencoder consists of 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 $\mathbf{X}$ [33]. In our framework, the functions f and g are deep convolutional and deconvolutional networks, respectively.

As shown in Fig. 4.4, the encoder function f consists of two sets of layers, where each set has two convolutional layers followed by a max pooling layer. The input to the encoder 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. 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) for the second set. 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

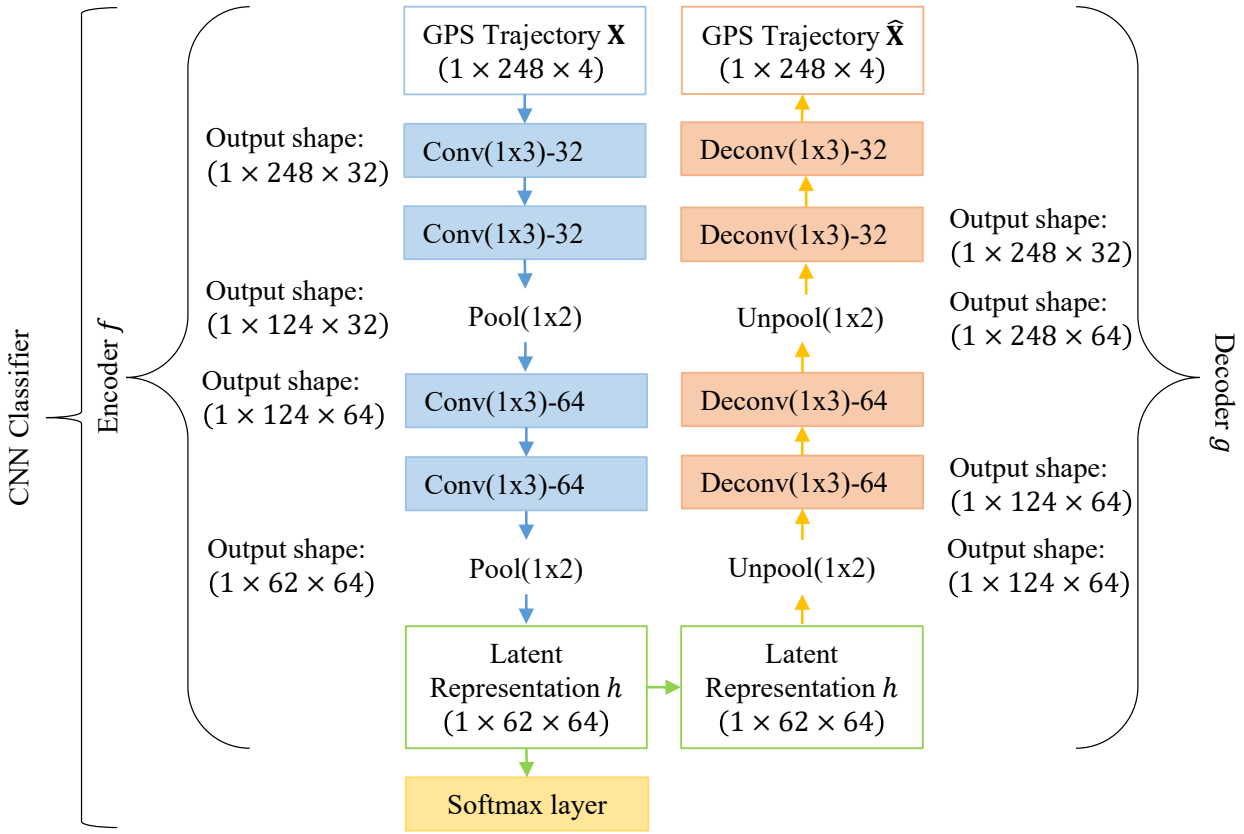


Figure 4.4: The architecture of our semi-supervised framework, which consists of the convolutional-deconvolutional autoencoder and CNN classifier. The layers' parameters are represented by "(filter size)-(number of filters)" for Conv. and Deconv. layers, and "(pooling size)" for pooling and unpooling layers. The "Output shape" denotes the output size of the corresponding layer, which is shown only when the output size changes.

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. The convolved neurons are activated by the Rectified Linear Unit ($ReLU$) function. 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 experimental observation, collapsing the latent representation h with 3 dimensions into a 1-dimension vector deteriorates the performance of the model. The tensor h in Fig. 4.4, for instance, has the shape size $h \in \mathbb{R}^{(1\times62\times64)}$.

The decoder function g has the same number of layers as the encoder and performs

the inverse operations (i.e., unpooling and deconvolutional) so as to generate an output with the same size of the input in the corresponding layer in the encoder part. For example, in Fig. 4.4, the tensor $h \in \mathbb{R}^{(1\times 62\times 64)}$ is first passed into the unpooling layer, which generates a tensor with the size $(1 \times 124 \times 64)$. Afterwards, the output feature map from the unpooling layer is fed into a deconvolutional layer, which results in a tensor with the same shape $(1 \times 124 \times 64)$. Except for the last layer, the activation function for all deconvolutional layers is $ReLU$. Proceeding with 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.

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:

$$l^{Conv-AE} = \sum_{i} (\hat{x}_i - x_i)^2 \tag{4.8}$$

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.

4.4.3.2 CNN-based 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. Therefore, the flattened latent representation h is directly fed into a $softmax$ layer to generate a probability distribution over the transportation labels for the GPS segment $\mathbf{X}_l$, denoted as $P_l = \{p_{l,1}, \cdots, p_{l,K}\}$, where K is the number of transportation modes. 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 widely accepted categorical cross-entropy is used as the loss function for the CNN classifier. The loss function for every labeled SE is formulated as follows:

$$l^{labeled-classifier} = -\sum_{i=1}^{K} y_{l,i} \log(p_{l,i}) \tag{4.9}$$

where $y_{l,i} \in \mathbf{Y}_l$ is a binary indicator which is equal to 1 if the class i is the true transportation label for the sample $\mathbf{X}_l$ and 0, otherwise. $\mathbf{Y}_l$ is the true label for $\mathbf{X}_l$, represented as one-hot encoding. Analogous to the Conv-AE, the cross entropy loss is averaged across the training batch in each iteration.

4.4.3.3 Model Training Our main training strategy is to simultaneously train the Conv-AE and CNN classifier. 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. 4.5, 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 pooling 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. (4.8) and (4.9):

$$l^{semi-ae+cls} = \alpha l^{Conv-AE} + \beta l^{labeled-classifier} \tag{4.10}$$

where α and β are the model hyperparameters that make a balance between the relative importance of the two losses in Eqs. (4.8) and (4.9). 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 [46], Glorot uniform initializer for initializing the layers weights [30], and a dropout regularization with the dropout ratio 0.5 before the $softmax$ layer to overcome the overfitting problem [78].

4.4.4 Parameter Tuning and Scheduling

Generally, two high-level tactics are applicable for scheduling α and β over training epochs: (1) Placing 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 the following epochs. The key motivation behind this strategy is to first learn the

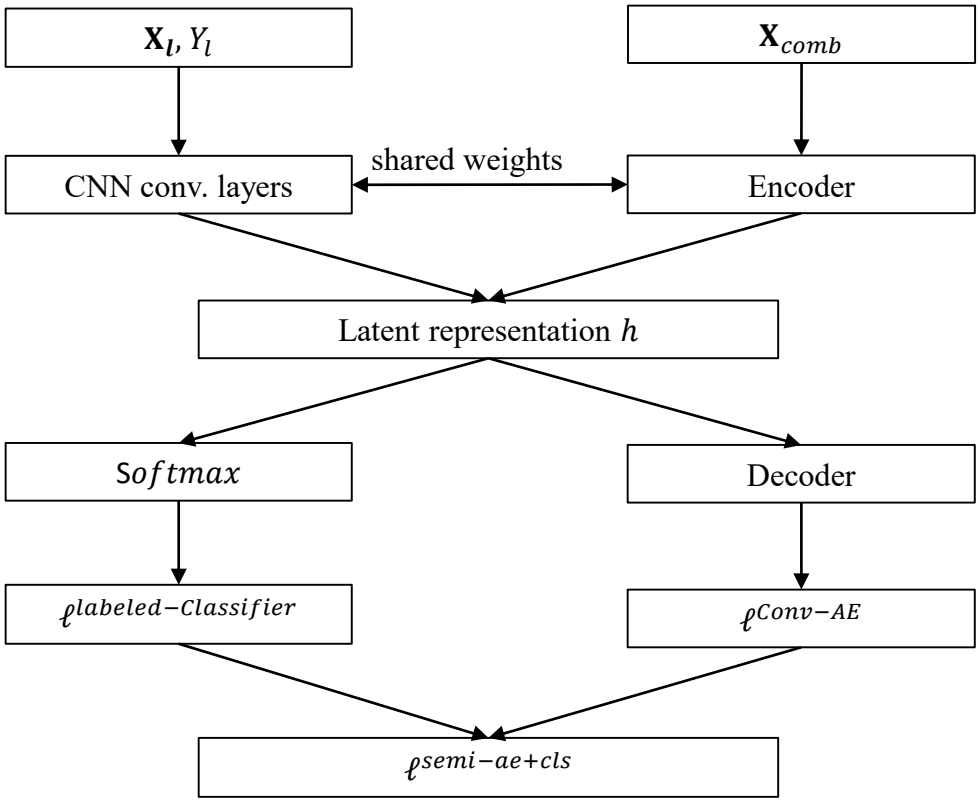


Figure 4.5: Flow for jointly training the supervised and unsupervised components of the proposed SECA model, depicted in Fig. 4.4

high-level features of GPS trajectories and next refine the learned features to be more discriminating for the classification task [95]; (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 [34].

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 marginally increased. 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 the 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 the following three main aspects.

- 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 [95].

- Unlike other studies [76, 95], 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 [63, 76, 95], balancing parameters are considered for both unsupervised and supervised components.

4.5 Experimental Results

In this section, we will evaluate the performance of our two-step trip segmentation and SECA model on large-scale GPS trajectory data. First the dataset along with the data cleaning and preparation steps are discussed. Then, various supervised and semi-supervised baseline models that are used for performance comparison are described. The prediction performance of our model is evaluated using widely-used classification metrics. The proposed training strategy and the overall model architecture are also evaluated against several alternative approaches.

4.5.1 Experimental Setup

4.5.1.1 Dataset Description and Data Pre-processing The proposed model is examined and validated on the GPS trajectories collected by 182 users in the GeoLife project. The raw dataset contains 17,621 trajectories with a total distance of 1,292,951 kilometers and a total duration of 50,176 hours [99]. To the best of our knowledge, this is the only public dataset with GPS trajectories that have also been annotated with transportation modes. 69 users have labeled their trajectories with transportation modes while the remaining users left their trajectories unlabeled. It is worth noting that not all trajectories of those 69 users were annotated, and the trajectories for which the annotation was missing were considered to be unlabeled data. Although many kinds of transport modes have been labeled by the users, only transport modes that constitute significant portion of the dataset are considered for our analysis. Our transportation mode list is $Y = \{$walk, bike, bus, driving, train$\}$. The time-interval threshold for dividing a user's GPS trajectory T into trips and the maximum number of GPS points in a SE (i.e., M) are set to 20 minutes and 248, respectively.

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

- A GPS point with the timestamp greater than its next GPS point is identified and discarded.

Table 4.1: 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

- 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 4.1, 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 1.5 times (or more) the interquartile range either above the third quartile or below the first quartile is identified and discarded.

- 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. In this study, these thresholds are set to 20, 150 meters and 1 minute, respectively.

The maximum allowable speed and acceleration pertaining to each mode are provided in Table 4.1, which have been defined using several reliable online sources and the engineering justification (e.g., existing speed limits, current vehicle/human's power).

Using the above-mentioned settings and the true change points, the distribution of the 4-channel labeled SE among various modes and the number of 4-channel unlabeled SE are listed in Table 4.1.

4.5.1.2 Baseline Methods We compare the performance of the proposed model with two sets of baseline methods: (1) supervised algorithms, and (2) semi-supervised algorithms.

With respect to the *supervised group*, widely used standard supervised algorithms in the literature of transportation mode detection are deployed for comparison, including K-Nearest Neighbors (KNN), RBF-based Support Vector Machine (SVM), Decision Tree (DT), and Multilayer Perceptron (MLP). The hand-crafted features introduced in [97, 98] are passed into these supervised algorithms, as the most acceptable manual GPS trajectories' attributes available in the literature. These features include the GPS segment's total distance, mean speed, expectation of speed, variance of speed, top three speeds, top three accelerations, heading change rate, stop rate, and speed change rate. After calculating motion features of every $p \in SE$ using Eqs. (1)-(5), these manual-designed features can be simply computed for a GPS SE using the definitions provided in [97, 98].

In addition, two supervised deep-learning models are also used as baselines: (1) CNN classifier with the same settings as in the proposed model, (2) Recurrent Neural Networks (RNN) with the long short-term memory (LSTM) module. The number of repeating modules is equal to the length of the 4-channel tensor (i.e., M) while the current input for each module is a feature vector corresponding to every GPS point p, where the feature vector contains RD_p, S_p, A_p, and J_p. According to the hyperparameter tuning analysis, the combination of one LSTM layer with 50 units in each LSTM module yields the best RNN performance. RNN is an important baseline since it has widely been used for modeling trajectory data in recent years [88, 94].

With regards to the *semi-supervised group*, two distinct baselines are used: Semi-Two-Steps and Semi-Pseudo-Label, which are categorized as two-step and joint training techniques, respectively, as described in Section 4.2.

- **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 using a standard supervised algorithm, which is a logistic regression (i.e., the $softmax$ layer) in our case. The loss functions for the

Conv-AE and logistic regression are given in Eqs. (4.8) and (4.9), respectively.

- **Semi-Pseudo-Label**: 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:

$$\begin{aligned} l^{semi-pseudo} &= \alpha l^{pseudo-classifier} + \beta l^{labeled-classifier} \\ l^{pseudo-classifier} &= -\sum_{i=1}^{K} y_{ul,i} \log(p_{ul,i}) \end{aligned} \tag{4.11}$$

  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 our SECA model, α and β are the balancing parameters. Further details about this approach is available in [49].

4.5.1.3 Performance Evaluation

The performance of our proposed trip segmentation is measured using precision and recall metrics, which are defined as below in the context of change point detection:

$$Precision = \frac{TP}{|\widehat{CP}|} \text{ and } Recall = \frac{TP}{|CP|}$$

where $|\widehat{CP}|$ and $|CP|$ are the number of predicted and true change points related to a GPS SE. TP is the number of true positives. A true change point is considered as a true positive if a predicted change point is found within a certain margin from the true change point. We set the margin to 20 GPS points, which is the same as the minimum of number allowed GPS points in a SE.

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

- *Accuracy* - it is computed as the fraction of SEs in the test set that are correctly classified.

- *Weighted F-measure* - F-measure for every transportation mode is defined as the harmonic average of its precision and recall in the test set, as shown below:

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

  Weighted F-measure is the weighted average of F-measure for each mode, in which the corresponding weight for each mode is the proportion of SE from that mode in the test set. For each transportation mode y, precision shows the fraction of the true SEs with the label y among all SEs that are classified as y, while recall implies the ability of the model to correctly identify SEs with the true label y.

In all our 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. However, unless stated, the entire unlabeled data are used for training in semi-supervised models. Using stratified sampling, 10% of the labeled training data in each fold is selected as the validation set for the early-stopping procedure used within the deep learning-based models. All the described data processing and models are implemented within Python programming environment using `ruptures` for the trip segmentation, `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

4.5.2 Performance Comparison Results

First, the performance of our SECA model is assessed without taking the segmentation process into account. In other words, the labeled GPS SEs are created based on the true change points rather than the predicted ones. Next, our proposed trip segmentation is evaluated. Finally, the impact of the segmentation process on the overall performance is evaluated.

Table 4.2: Comparison of accuracy values for different supervised and semi-supervised models with varying amounts of labeled data. All unlabeled data are used for training.

	Proportion of labeled *SE* 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-Pseudo-Label	0.589 (±0.020)	0.663 (±0.023)	0.707 (±0.021)	0.733 (±0.021)	0.754 (±0.018)
SECA (ours)	0.629 (±0.010)	**0.693 (±0.019)**	**0.732 (±0.017)**	**0.750 (±0.016)**	**0.768 (±0.016)**

Table 4.3: Comparison of weighted F-measure values for various supervised and semi-supervised models with varying amounts of labeled data. All unlabeled data are used for training.

	Proportion of labeled *SE* in the training data				
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)
Supervised-RNN	0.318 (±0.044)	0.325 (±0.039)	0.336 (±0.032)	0.358 (±0.034)	0.369 (±0.032)
Semi-Two-Steps	0.512 (±0.026)	0.541 (±0.038)	0.574 (±0.016)	0.584 (±0.014)	0.589 (±0.019)
Semi-Pseudo-Label	0.582 (±0.020)	0.654 (±0.025)	0.701 (±0.022)	0.728 (±0.021)	0.749 (±0.019)
SECA (ours)	0.615 (±0.011)	**0.683 (±0.019)**	**0.725 (±0.017)**	**0.745 (±0.017)**	**0.764 (±0.017)**

4.5.2.1 SECA Evaluation Tables 4.2 and 4.3 provide the performance results of our SECA model and baseline methods in terms of accuracy and weighted F-measure, respectively. Every model is trained using various amounts of labeled data so as to investigate the effectiveness of the models when different amounts of labeled data is used.

Table 4.2 clearly shows the superiority of our SECA model and its training strategy in comparison with other 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. What is interesting about the result is the low-quality of MLP and RNN, which also indicates that employing deep learning architectures does not always result in a better performance compared to shallow structures. Comparison between CNN and RNN clearly shows that capturing the local correlation between adjacent GPS points by the convolution operation generates more efficient features in comparison with the attempt to learn the long-term dependency between all GPS points. In comparison between supervised and semi-supervised algorithms, the results indicate that semi-supervised techniques perform better than their supervised counterparts. Indeed, modeling the distribution of input data through unsupervised learning techniques with the help of unlabeled data can potentially ameliorate the generalization ability of the supervised task. Focusing on only semi-supervised algorithms, it is obvious that the prediction quality of joint training strategies (i.e., SECA and Semi-Pseudo-Label) are significantly higher than the Semi-Two-Steps. Such evidence confirms that, with aid of efficient initialization techniques, simultaneously training the reconstruction and classification abilities of a semi-supervised model yields a better performance compared to disjoint training strategies. For an overall comparison, our SECA model achieves on average 6.2% higher accuracy compared to other methods over different amounts labeled data, excluding the supervised algorithms (i.e., KNN, SVM, MLP, and RNN), which have not obtained competitive results in our problem.

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 4.3, weighted F-measure for our SECA model is also superior to all other algorithms. This evidently confirms our findings and reasoning based on results in Table 4.2.

4.5.2.2 Trip Segmentation Evaluation Figure 4.6 shows the performance of our two-step trip segmentation process in terms of precision and recall for different values of the penalty level γ. The results clearly indicate the effectiveness of our proposed approach by achieving the very high values of 0.99 and 0.93 for precision and recall, respectively. As expected, absence of the penalty function in Eq. 4.6 causes overfitting since it allows the search algorithm choosing more and more change points in order to decrease the first component in the right side of Eq. 4.6. Predicting a high number of false change points results in a low precision value. This issue can be controlled by increasing the value of γ in order to constrain the number of predicted change points. Interestingly, while the precision value is dramatically improving (i.e., almost closing to 1) by raising the γ value, the recall value is dropping down very slightly. The rationale behind such a behavior is that the number of true change points in each GPS SE is very low after implementing the first step of the trip segmentation process. Our analysis indicates that more than 99% of the GPS SE contains less than 2 change points after the first step segmentation, which also implies the importance of the first step. Therefore, using larger values of γ forces the search algorithm to predict a few yet correct number of change points, which in turn results in high value of both precision and recall.

4.5.2.3 Overall Performance Evaluation Table 4.4 shows the accuracy and F-measure results of our SECA model for three trip segmentation scenarios: (1) Trips are segmented based on the true change points. In other words, we assume the change points are already known, which leads to have the same result as in Tables 4.2 and 4.3, (2) Trips are uniformly partitioned into segments with the minimum number of GPS points, which is 20 in our setting. This scenario guarantees that all GPS SEs have only one transportation as SEs contains only the minimum allowed GPS points, (3) Trips are partitioned based on our proposed two-step trip segmentation process, which represents our whole mode detection framework (i.e., the SECA model followed by the results of the two-step trip segmentation). As can be seen from Table 4.4, the overall performance of our proposed framework is degraded by only 4% (on-average) compared to when the true change points

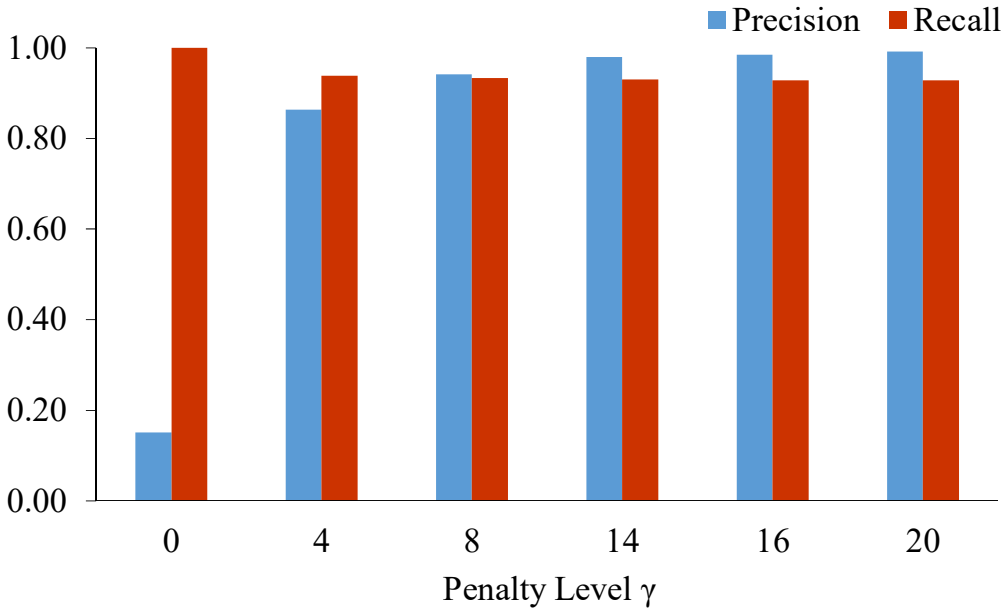


Figure 4.6: Precision and recall values for the proposed trip segmentation process with different values of the penalty level γ.

are known. This implicitly indicates the acceptable performance of our trip segmentation. However, naively partitioning trips into uniform-size segments with the minimum number of GPS points (i.e., the second scenario) leads to lose many GPS information in a segment and in turn making mode detection more difficult for the SECA model. The performance metrics for the second scenario decreases by on average 10% compared to the first scenario, which is 6% less than the overall performance of our proposed framework.

Table 4.4: Comparison of accuracy (Acc.) and F-measure (F1) for our SECA model while different trip segmentation scenarios are applied.

	Proportion of labeled *SE* in the training data									
	10%		**25%**		**50%**		**75%**		**100%**	
Trip Segmentation	**Acc.**	**F1**	**Acc.**	**F1**	**Acc.**	**F1**	**Acc.**	**F1**	**Acc.**	**F1**
True Change Points	0.629	0.615	0.693	0.683	0.732	0.725	0.750	0.745	0.768	0.764
Uniform Size	0.536	0.514	0.584	0.572	0.630	0.623	0.646	0.637	0.671	0.664
Two-step Segmentation (ours)	0.600	0.589	0.650	0.641	0.680	0.673	0.704	0.700	0.721	0.717

4.5.3 Analysis and Discussion

In this section, we assess the quality of our proposed framework in several other aspects including the hyperparameter schedule, the data structure for GPS SEs, the SECA architecture, and the prediction capability for every transportation mode.

4.5.3.1 Balancing Parameters Schedule We will now discuss the proposed schedule for tuning the balancing parameters α and β. Table 4.5 presents the test accuracy of the SECA model, 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 entire training process. Note that the schedules #1, #2, and #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 the training process. 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 4.5, 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 noting 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 4.2 and 4.3 supports the effectiveness of our proposed hyperparameter schedule. Note that Semi-Pseudo-Label is the most competitive technique to our approach and is jointly trained by varying its balancing parameters according to our proposed schedule.

Table 4.5: Comparison of accuracy values for different hyperparameter schedules along with different sizes of labeled data. $1 \rightarrow 0.1$: Gradually decreasing from 1 to 0.1 over training iterations.

	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 ($\pm$0.018)	0.675 ($\pm$0.015)	0.709 ($\pm$0.008)	0.712 ($\pm$0.020)	0.747 ($\pm$0.022)
2	$\alpha : 1$ $\beta : 1 \rightarrow 0.1$	NA	0.618 ($\pm$0.017)	0.670 ($\pm$0.024)	0.715 ($\pm$0.015)	0.721 ($\pm$0.017)	0.741 ($\pm$0.012)
3	$\alpha : 1$ $\beta : 1$	NA	0.625 ($\pm$0.018)	0.667 ($\pm$0.016)	0.716 ($\pm$0.015)	0.734 ($\pm$0.019)	0.745 ($\pm$0.022)
4	$\alpha : 1$ $\beta : 0$	$\alpha : 1$ $\beta : 1$	0.551 ($\pm$0.021)	0.564 ($\pm$0.214)	0.703 ($\pm$0.028)	0.715 ($\pm$0.023)	0.739 ($\pm$0.024)
5	$\alpha : 0$ $\beta : 1$	$\alpha : 1$ $\beta : 1$	0.590 ($\pm$0.023)	0.671 ($\pm$0.022)	0.711 ($\pm$0.025)	0.732 ($\pm$0.023)	0.755 ($\pm$0.022)
6 (ours)	$\alpha : 1$ $\beta : 1$	$\alpha : 1$ $\beta : 0.1$	**0.629 ($\pm$0.010)**	**0.693** ($\pm$0.019)	**0.732** ($\pm$0.017)	**0.750** ($\pm$0.016)	**0.768** ($\pm$0.016)

4.5.3.2 Feature Analysis for GPS *SE* Representation The quality of our proposed layout for representing motion features of a SE, shown in Fig. 4.3, is rigorously evaluated by tracking the effectiveness of various motion-feature combinations. Table 4.6 summarizes the performance comparisons when the input tensor is created using a single or a combination of features, as described in Section 4.3.2. The number of motion features determines the number of channels in the tensor. For example, an input tensor with only S information contains one channel rather than four. First, each feature type is independently examined in order to detect the most salient feature types, which in turn helps in constructing better feature combinations. As observed in Table 4.6, speed (S), relative distance (RD), and acceleration (A) are the most effective features when used in a stand-alone setting. The information obtained from tensors with single features leads to examine more reliable feature combinations. The combinations of the first two, the first three, and the first four important features (i.e., $RD+S$, $RD+S+A$, $RD+\Delta t+S+A$, respectively) are the most reasonable initial selections. A 6-channel tensor with all feature types is another selection in Table 4.6. Nonetheless, our configuration, that fuses relative distance, speed, acceleration, and jerk, attains the best performance compared to other potential good configurations. We also replace the jerk with bearing rate (i.e., $RD+S+A+BR$) so as to integrate the heading direction with kinematic information, but no further improvement is achieved. It is worth noting that Table 4.6 encapsulates the information for a few yet potential combinations

that have attained the highest accuracy.

Table 4.6: Comparison of accuracy and weighted F-measure for various feature combinations.

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

4.5.3.3 Analysis of Model Architecture The depth of neural networks is a key hyperparameter in deep architectures. Thus, the structure of our SECA model is evaluated in terms of depth by steadily increasing its depth through adding convolutional layers. Analogous to the SECA structure, depicted in Fig. 4.4, the number of filters starts with 32 for the first two convolutional layers and increases by a factor of 2 after adding every two convolutional layers. Every two convolutional layers are followed by a max-pooling layer. Other parameters (e.g., filter size) are fixed throughout the network. Table 4.7 shows the average accuracy values in 5-fold cross-validation for our SECA model with 2, 4, 6, and 8 convolutional layers by varying amounts of labeled SE. The last column is the average over all amounts of labeled data. It can be seen that increasing the number of layers up to the 4 layers enhances the model accuracy by around 1%, whereas adding additional layers does not result in a substantial improvement. Accordingly, we stop at a model with 4 convolutional layers (i.e., our proposed SECA model) so as to reduce the computation time.

Table 4.7: Evaluation of the model configuration of the proposed SECA method by varying the number of convolutional layers across different amounts of labeled SE in the training data.

No.	**% labeled *SE* in the training data**					
Layers	**10%**	**25%**	**50%**	**75%**	**100%**	**Average**
2	0.621	0.680	0.723	0.741	0.751	0.703
4	0.629	0.693	0.732	0.750	0.768	**0.714**
6	0.629	0.693	0.732	0.750	0.759	0.712
8	0.629	0.698	0.725	0.743	0.762	0.711

4.5.3.4 Prediction Capability Per Transport Mode As our last round of evaluation, we delve into the confusion matrix to analyze the high-level prediction ability of our SECA model for every transportation mode. Table 4.8 illustrates the confusion matrix, as well as precision and recall pertaining to each transportation mode for a test set. As expected, there is a high correlation between prediction quality and the number of available SE for a mode in the training set. As shown in Table 4.1, the walk and driving modes constitute the largest and smallest portions of the GPS SEs, which in turn results in the best and worst prediction performance for walk and driving modes, respectively. Nonetheless, the discriminating moving pattern of walk compared to others is another principal reason in achieving a perfect recall value for the walk mode. In addition to the lack of labeled driving SEs, the poor performance of the model in estimating the driving mode stems from several other factors including the possibility for driving in alternative routes with different speed limits, the presence of various types of drivers' behavior, and the flexibility of drivers in maneuvering. On the other hand, bus and train are the transit modes that must adhere to pre-defined routes and schedule, which leads to more predictable mobility behavior. Another interesting yet reasonable finding is that a large portion of false negative SEs for the driving mode is bus since bus is the most identical mode to car and taxi. Analogously, a majority of bike SEs has been falsely classified as walk as the moving pattern of bike is closer to walk compared to other modes. Such misclassifications calls for more labeled training SE so as to improve the discrimination ability of our SECA model.

Table 4.8: Confusion matrix for our SECA model. Prec. and Rec. correspond to Precision and Recall, respectively.

		Predicted Modes					
		Walk	**Bike**	**Bus**	**Drive**	**Train**	**Rec.**
True Modes	**Walk**	1269	43	16	0	0	0.96
	Bike	110	577	66	6	3	0.75
	Bus	173	41	927	49	20	0.76
	Drive	95	27	159	533	51	0.61
	Train	57	13	60	58	469	0.71
	Prec.	0.74	0.82	0.75	0.82	0.86	

4.6 Conclusion

We proposed a two-step trip segmentation and semi-supervised convolutional autoencoder (SECA) model for identifying transportation modes from GPS trajectory data. First, the raw GPS trajectory of a user's trip was partitioned into GPS segments with only one transportation mode. Next, every GPS segment was converted into an efficient 4-channel tensor so as to utilize the convolutional operation for automatically extracting high-level features rather than using handcrafted features. The unsupervised and supervised components of the SECA model were simultaneously trained on both unlabeled and labeled GPS tensors while an optimal schedule was deployed for varying the balancing parameters between reconstruction and classification losses. Our extensive experiments demonstrated the superiority of the proposed trip segmentation process, the SECA model, the hyperparameter schedule, the representation for GPS trajectories, and the configuration of the model architecture compared to several baselines and alternatives.

Chapter 5

Developing a Twitter-Based Traffic Event Detection Model Using Deep Learning Architectures

Abstract

In recent years, several studies have harnessed Twitter data for detecting traffic incidents and monitoring traffic conditions. Researchers have utilized the bag-of-words representation for converting tweets into numerical feature vectors. However, the bag-of-words not only ignores the order of tweet's words but suffers from the curse of dimensionality and sparsity. A common approach in literature for dimensionality reduction is to build the bag-of-words on the top of pre-defined traffic keywords. The immediate criticisms to such a strategy are that the pre-defined set of keywords may not include all traffic keywords and the tweet language is subjected to change over time. To address these shortcomings, we utilize the power of deep-learning architectures for both representing tweets in numerical vectors and classifying them into three categories: 1) non-traffic, 2) traffic incident, and 3) traffic information and condition. First, we map tweets into low-dimensional vector space through word-embedding tools, which are also capable of measuring the semantic relationship between words. Supervised deep-learning algorithms including convolutional neural network (CNN) and recurrent neural network (RNN) are then deployed on the top of word-embedding models for detecting traffic events. For training and testing our proposed model, a large volume of traffic tweets is collected through Twitter API endpoints and labeled through an efficient strategy. Experimental results on our labeled dataset show that the proposed approach achieves clear improvements over state-of-the-art methods.

Keywords: Deep learning; Twitter; Recurrent and Convolutional Neural Net-

works; Traffic information systems

5.1 Introduction

The substantial increase in traffic demand in urban and rural areas leads to traffic congestion, which calls for developing traffic control and management strategies to mitigate both recurring and non-recurring congestion. Recurring congestion refers to predictable conditions when demand exceeds transportation system's capacity such as daily rush hours. Non-recurring congestion is often caused by unpredictable incidents such as crashes, disabled vehicles, roadway debris, and weather-related issues. Providing traffic condition information for both drivers and traffic managers in a real-time and efficient manner is of utmost importance. Online traffic information helps travelers choose the shortest and/or fastest driving routes using their navigation systems. Also, Traffic Incident Management (TIM) partners, such as transportation agencies and state police, can promptly recognize unexpected behavior in traffic flow characteristics, which helps restore smooth traffic flow as quickly and safely as possible.

Historically, traffic flow sensors (e.g., CCTV cameras and inductive loop detectors) and probe vehicles equipped with positioning tools (e.g., GPS) have been widely used as the main sources for collecting traffic information including lane occupancy, speed, vehicle counts, and travel time. Simple analysis of the collected data can identify the current road traffic congestion level. For example, by applying the mathematical algorithms on these measurements, unexpected traffic flow characteristics in both downstream and upstream of the suspected location can be identified and labeled as the occurrence of a traffic incident. However, the results of nationwide surveys from professional practitioners revealed the failure of such incident algorithms, particularly regarding unacceptable false-alarm rates and a long detection time [59, 87]. Although CCTV-based visual detections by traffic operators and mobile-phone calls from motorists may provide reasonable detection functionality in some situations, both the rapid growth of transportation networks and labor costs hinder having a cost-effective and reliable incident detection scheme using only human-based methods [87]. To mitigate these deficiencies, crowdsourcing using social network users can be involved as a complementary traffic data

source.

The growth of social networking tools has been generating a massive data source that contains rich information about traffic conditions. Waze is a successful example of crowdsourcing navigation systems that also allows users to report on traffic, accidents, police traps, blocked roads, and weather conditions. Collected information is then disseminated to other Wazers on a real-time basis. However, in terms of communication, Wazers are unable to report a traffic condition that does not fall into one of the pre-determined incident categories in Waze [35]. Furthermore, since Waze has been designed only for private cars, no information is shared about the traffic network sections that have been dedicated to public transportation, bicycles or trucks. Finally, Waze data are only accessible to approved partners, which in turn makes accessing to traffic data difficult for public agencies (e.g., traffic management centers). Twitter can be a viable alternative for addressing the shortcomings in Waze.

Twitter is one of the most popular microblogging services worldwide with more than 330 million active users, according to Statista Portal. Twitter enables users to post and read short messages, called tweets, about 'what is happening now'. Real-life events, which occur over the space and time, can be reported by users through the Twitter platform. The limitation on the tweet length, which was originally 140 characters per tweet and recently expanded to 280 characters, forces people to communicate in a timely and effective fashion. For instance, user-generated tweet texts about a traffic crash can be viewed as witnesses' reports, which is similar to traditional drivers' phone calls. Event detection from Twitter streams has received an increased interest in the past decade [2]. Monitoring and analyzing the flow of tweet texts results in detecting real-world events occurring at a specific time and place. This paper attempts to extract real-time traffic congestion and incident events using Twitter data, as a complementary traffic information resource.

Unlike the existing fixed-point traffic flow sensors, Twitter data are not limited by sparse coverage and can be considered as an efficient and ubiquitous data source for traffic operation and incident management systems [18]. Therefore, extracting traffic information from Twitter text paves the way for generating an inexpensive, real time, and widespread traffic monitoring system. Since every out-of-home activity relies on transportation systems, people as dynamic social sensors might

immediately report traffic conditions (e.g., slow traffic) and traffic incidents (e.g., accidents or crashes) by posting tweets while they are driving or observing the occurrence of a traffic-related event. By default, users' tweets are available to public. Twitter allows developers to access such public data streams through its application programming interfaces (APIs). Extracting traffic-related tweets on a real-time basis leads to be aware of traffic conditions sooner than other traffic resources such as news channels. Our main objective is to detect the tweets that either describe a traffic-related incident or give updates on the status of traffic networks.

To discriminate traffic-related tweets from non-traffic-related tweets, several studies have exploited text mining and machine learning techniques to automatically extract meaningful information from unstructured tweet texts. Before applying machine learning techniques on text documents, tweet texts with variable number of words need to be mapped into fixed-size numerical feature vectors. The bag-of-words method is the most common way of text representation. In this method, tweets are split into a group of tokens (e.g., words) and a unique integer id is assigned to each possible token. So, each tweet is represented by an N-dimensional feature vector, where N is the number of distinct tokens in the tweet collection. The feature value corresponding to each token in the vector can be binary occurrence or a function of token-occurrence frequency (e.g., the classical term frequency-inverse document frequency tf-idf). Concatenating all tweets' vectors creates a feature space matrix with one row per tweet and one column per distinct token in the tweet corpus. Next, this matrix is fed into a supervising learning algorithm to perform the classification task.

However, the bag-of-words representation is imperiled by some challenges. First, it completely ignores the temporal order of words, which leads to missing the semantic and syntactic relationship between words in a tweet. Secondly, as the number of possible words in the tweet corpus is very large and only a small subset of words is used in each tweet, the resulting matrix suffers from sparsity with many features values as zero and the curse of dimensionality. To address the latter problem, feature selection methods are deployed to keep only high-importance features and dump the rest. Nonetheless, statistical models for selecting features from a sparse and high-dimensional matrix might not take the domain knowledge of the subject

application into account. Research has offered techniques to model tweets on the top of certain features with a high-occurrence frequency in traffic-related tweets. In other words, one can first build a vocabulary that consists of keywords associated with traffic conditions and then generates the bag-of-words representation in which features are the keywords in the built vocabulary. Although such a strategy engenders a much-lower-dimensional matrix, the immediate criticism to use a pre-defined set of keywords as features is that the vocabulary may not include all important traffic-related keywords and is subjected to alter over time. New users, who their samples of tweets are not in the training set, might use different words to describe traffic conditions. Also, the twitter language contains many informal, irregular, and abbreviated words as well as a large number of spelling and grammatical errors [2]. Thus, representing tweets on the top of a pre-defined set of words (i.e., features) might not build a sophisticated classifier that is robust against tweet diversity from various datasets.

Addressing the above-mentioned issues, we exploit unsupervised deep learning algorithms for tweet modeling and supervised deep learning architectures for the classification task. Word-embedding models are utilized to not only represent tweet texts in low-dimensional space but take the semantic relationship between tweet words into account. Recurrent neural networks (RNN) and convolutional neural networks (CNN) are then utilized for the classification task by extracting high-level and efficient features from distributed representations of words in each tweet. In particular, RNN and CNN are used to learn long-term dependencies between tweet words and capture local correlation between consecutive words, respectively. Since deep-learning architectures require a large volume of training samples to efficiently learn patterns and variations inside the data, we collect and annotate more than 50,000 tweets into three labels: non-related to traffic, traffic incidents, traffic condition and information. The two latter labels can be categorized as the traffic-related tweet. Some tweaks to increase efficiency are deployed to expedite our labeling process while preserving the quality of labeled data as high as possible. To the best of our knowledge, most previous studies have collected less than 5,000 labeled tweets for developing their Twitter-based traffic information model, without implementing any adjustments for obtaining fast-labeling results. Furthermore, since no labeled dataset on traffic-related tweets is publicly available, we release our collected dataset as a possible benchmark for future studies.

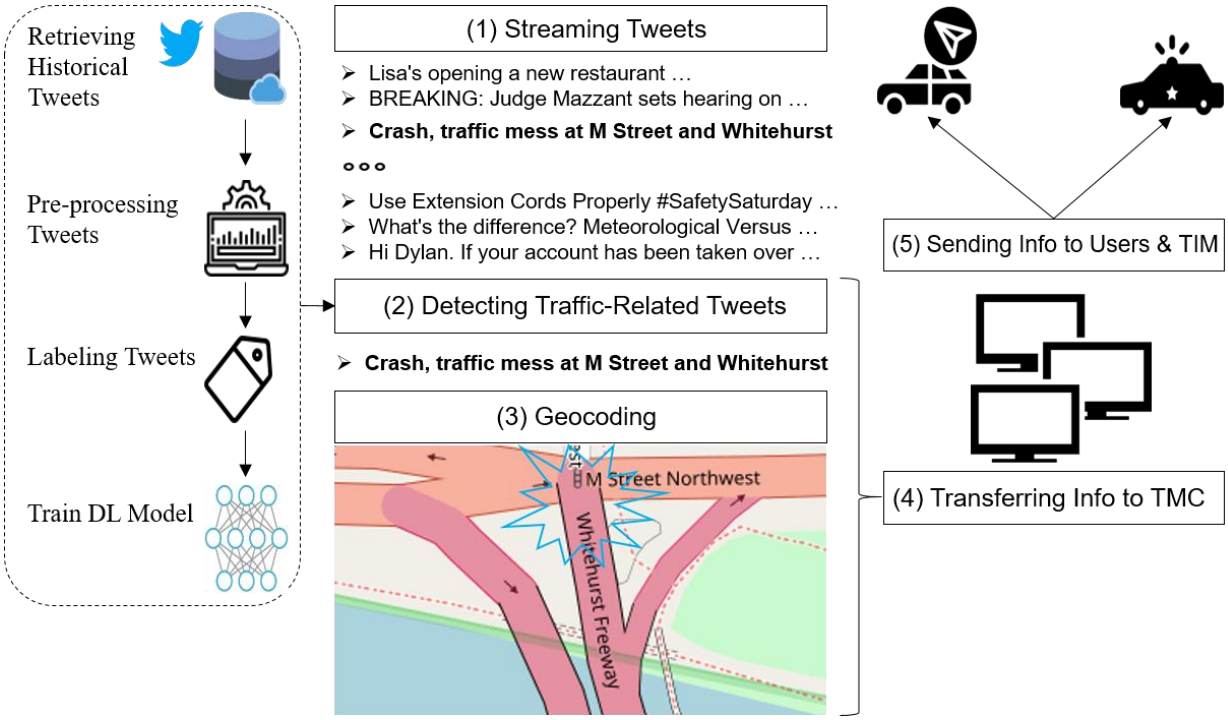


Figure 5.1: Fig. 1. Overview of a Twitter-based traffic information system. In the online operation, traffic-related tweets are extracted from streaming tweets using a Deep-Learning (DL) model trained on labeled, historical tweets. The detected tweets and their geocoded locations are then transferred to Traffic Management Center (TMC). Finally, approved information by TMC is passed to traffic users and Traffic Incident Management (TIM) partners.

After obtaining traffic-related tweets, the last step is to identify the location of the occurred traffic event. Unfortunately, a majority of generated tweets does not contain the coordinates attribute, which represents the geographic location of the tweet when posted. Even if the location service is turned on in a device, the coordinates attribute may not necessarily indicate the event location. For example, a user might post the relevant tweet when they arrive in a safe place. However, when a user reports a traffic condition, they often provide partial information about the location such as road names, exit numbers, and mile markers. Accordingly, the most reliable option for identifying the event location is to geocode the partial information that is typically available inside the tweet text. Geocoding is the process of translating a physical address description into its corresponding geographic coordinates. Nonetheless, the irregularity and incompleteness of the location information makes the geocoding problem a challenge.

Figure 5.1 depicts the overview of a Twitter-based traffic information system. When a traffic event occurs, some travelers might describe the traffic incident or the network condition on a real-time basis. In the meantime, a trained classifier extracts the traffic-related tweets from real-time tweets that are accessible through Twitter streaming APIs. Next, the traffic-related tweets are fed into a geocoder to obtain the location information of the occurred event. Finally, the location information is transferred to traffic management centers (TMC). Upon the approval of the event occurrence by TMC based on information from other resources, the location information and other details of the detected event are disseminated to transportation users and TIM partners so as to respond appropriately. The assessment of the geocoder and the whole system requires both historical tweet data and the external ground truth (e.g., historical incident reports) for the same period. Due to lack of such data sources, the scope of this study is limited to collecting tweets and building a model for detecting traffic-related tweets. The key contributions of this work are summarized as follows:

- **Collecting and labeling a large volume of tweets.** More than 50,000 tweets are collected through Twitter APIs and labeled into three classes. Some shortcuts to increase efficiency are implemented in the labeling approach in order to increase speed without sacrificing the accuracy and quality of the labeled data. To the best of our knowledge, the collected dataset is by far the largest labeled-tweet dataset in the traffic domain.

- **Developing a traffic event detection model using deep-learning architectures**. For the first time in this domain (i.e., traffic event detection from Twitter data), tweets are modeled into numerical feature vectors using word-embedding tools. Next, CNN and RNN are applied to tweet word-embedding matrixes for discriminating traffic-related tweets from non-traffic tweets.

- **Conducting an extensive set of experiments for performance evaluation and comparison.** First, proposed models in relevant studies are reproduced and used as the baselines. Comparison results clearly reveal that our proposed model outperforms the state-of-the-art models for detecting traffic-related tweets from Twitter data. Furthermore, we demonstrate

the superiority of model settings and its capability for real-world situations through several other experiments.

The rest of this article is organized as follows. After reviewing related works in Section 2, the process of collecting and labeling tweets is set out in Section 3. The details of our proposed framework are elaborated in Section 4. In Section 5, several experiments are conducted to evaluate the performance of the proposed framework. Finally, the paper is concluded in Section 6.

5.2 Related works

In recent years, a considerable literature has emerged around the theme of event detection from Twitter streams. The event topics ranges from sports [1], politics [3], and disease activities [73] to natural disasters [69] and traffic incidents [35]. Furthermore, Twitter streams can also be monitored to identify unknown events such as breaking news [61]. A common approach for detecting unspecified events is to cluster temporal patterns using the highly frequent keywords that suddenly burst Twitter. A comprehensive survey on event detection techniques from Twitter data is available in [2]. The article categorizes the techniques according to event type, detection task, detection method, and the target application. Since this paper seeks for designing a type of traffic monitoring system using tweets, this section primarily focuses on the studies that traffic-event detection from Twitter constitutes the major part of their proposed framework.

A tweet object consists of many root-level attributes including users' profile, geographic location of the tweet, creation time, place associated with tweets, and text. Although some studies have used non-text attributes for their analysis [12], the tweet text has been regarded as the most salient feature. Hence, much of the previous research on Twitter data has carried out natural language processing (NLP) and text mining techniques so as to monitor traffic conditions and detect traffic incidents. In these studies, a procedure upon tweet texts for discriminating traffic-related tweets has been proposed. Table 5.1 encapsulates a majority of studies that quantitatively analyzed tweet texts to gleam traffic information. These studies can be further categorized into two sub groups based on how a tweet text

is represented: 1) Non-numerical features. In this group, tweets are represented based on whether a tweet contains a pre-defined set of keywords. 2) Numerical features. Unlike the first group, tweets are mapped to numerical feature vectors, typically using the bag-of-words representation. For each work, the methods for representing texts and classifying tweets as well as the number of labeled tweets for training and testing are shown in Table 5.1.

One of the earliest works from the first category was conducted by Wanichayapong et al. [86]. The key goal in their proposed model was to differentiate between traffic-point and traffic-link tweets. The former associates with only one point (e.g., a crash) while the latter type associates with a road-start point and a road-end point (e.g., traffic jam). A tweet is tokenized and parsed based on four categories of words: 1) places such as roads, 2) traffic problems such as accident, 3) words indicating start and end locations of traffic events, 4) ban words (i.e., vulgarity/profanity words). Since tweets are not represented with numerical features, rule-based and heuristic methods are applied to first filter traffic-related tweets and then classify traffic tweets into point and link types. All heuristics are adopted based on what types of word categories a tweet contains. Another example of the first category is the work by Rebelo et al. [65], who utilized different variants of Pearson Correlation Coefficients to measure the correlation of traffic volumes and locations between desired tweets and ground-truth tweets. The selected tweets are those that contain traffic-related keywords. Ribeiro Jr et al. [67] developed their Traffic Observatory framework to detect traffic conditions and traffic events using Twitter data, geocode them, and display them on the Web. Traffic-related tweets were manually identified by searching a static and predefined list of words (e.g., intense, slow, regular, and free) that express traffic situations. However, the focus of the study was to locate traffic-related tweets by performing both exact and fuzzy matching using an enhanced gazetteer that contains urban details.

With regard to the numerical features category, Carvalho et al. [11] was one of the earliest studies that tokenized tweets into uni-grams and bi-grams as the feature representation. Support Vector Machine (SVM) was deployed to infer traffic-related tweets. In a more systematic analysis by D'Andrea et al. [19], after applying some preprocessing steps (e.g., stop-word filtering and word stemming), the tweet collection was transformed to a set of numeric vectors using the bag-of-words rep-

Table 5.1: Twitter-based traffic information systems in literature. NA: Not Available

Reference	Tweet representation	Classification method	No. of labeled tweets
Wanichayapong et al. (2011)	Non-numerical features	Rule-based and heuristic methods	6,253
Rebelo, et al. (2015)	Non-numerical features	Correlation coefficients	NA
Ribeiro Jr et al. (2012)	Non-numerical features	Manually	505
Carvalho (2010)	Numerical features	Support Vector Machine	4,092
D'Andrea, et al. (2015)	Numerical features	Support Vector Machine	1,330
Fu, et al. (2015)	Numerical features	Ranking based on feature weights	3,200
Gu, et al. (2016)	Numerical features	Semi Naïve Bayes	22,200
Pereira, et al. (2017)	Numerical features	Support Vector Machine	5,000

resentation. Then, a set of relevant features were extracted using Information Gain based on entropy, which resulted in converting the initial sparse bag-of-words matrix into a dense matrix. Finally, using supervised learning algorithms such as SVM, tweets were classified into three categories: non-traffic, traffic due to congestion, and traffic due to external events. In a study conducted by [29], a tweet-based framework was offered to recognize traffic-related tweets. Their approach comprised three parts: 1) building traffic-incident-related keywords using historical tweets of influential users. The built vocabulary consists of words with the highest tf-idf index, 2) crawl the real-time tweets that include a combination of a few keywords, called wordset. The frequent wordsets are identified by applying the Apriori algorithm to the vocabulary set, 3) rank the retrieved tweets based on the tf-idf weights of the contained keywords and select the most traffic-related tweets. In a follow-up and more comprehensive study by Gu et al. [35], an iterative data acquisition process for building a dictionary with traffic-related words was suggested. The words in the built dictionary were served as features in the Semi-Naïve-Bayes method to classify tweets into traffic incident versus non-traffic incident groups. Afterward, the TI tweets are geocoded and classified into five incident categories including accidents, road work, hazards weather, events, and obstacle vehicles using a Supervised Latent Dirichlet Allocation. A recent study by Pereira et al. [60] integrated the bag-of-words and word embedding to create a group of features for each tweet. The features are then fed into traditional classifiers to infer travel-related tweets.

With regard to Table 5.1, no study has mapped tweets into numerical features purely based on distributed vector representations of words, which dramatically reduces vector space dimensionality without neither of using statistical feature

selection methods nor demanding for a set of traffic keywords. In terms of classification, none of the mentioned studies have utilized CNN and RNN architectures, which are capable of capturing local correlation and long-term dependencies between tweet's words. Accordingly, we aim to fill this gap by leveraging an end-to-end deep learning framework for both tweet modeling and classification, which have not been implemented in this domain yet. Note that deep learning architectures have recently been attracting a lot of interest in various transportation fields [17].

5.3 Data collection and labeling procedure

As one of the contributions in this study, we have collected 51,100 tweets using the Twitter APIs and labeled them into three classes. Since labeling data is an expensive and time-consuming task, we propose an efficient labeling approach to obtain fast results with high quality. The approach is a form of internal labeling, in which labelers are the authors of this study. Internal labeling gives room to track the labeling progress and ensures the highest possible labeling accuracy compared to alternative procedures such as outsourcing, crowdsourcing, and data programming; however, it takes more time. Our proposed approach expedites the labeling process while not sacrificing the accuracy and quality of labeled data. Before proceeding to the details of the data collection and annotation, the Twitter API endpoints for returning tweets and the label definitions for the purpose of our application are described.

5.3.1 Twitter API

Twitter has several API methods for engaging with tweets, users, direct messages, and places. The three major tiers of Twitter APIs are standard, premium, and enterprise. The standard API family, used in this study, is a free-of-charge yet limited service while the other two provide complete yet paid access to Twitter objects dating all the way back to the first Tweet in March 2006. Twitter contains several objects such as Tweet and User, which are the responses to API methods. The Tweet object is the basic atomic building block in Twitter that consists of var-

ious attributes. The fundamental attributes used in this study include created_at (UCT time when the Tweet is created), id_str (the unique identifier of the Tweet), user (the User object who posted the Tweet), text (the actual text of the Tweet), retweeted_status (the original Tweet that was retweeted by the current Tweet). The User object provides information on public Twitter account and described the generator of a Tweet object. The User object contains a wide range of metadata, but the screen_name (a unique name for identifying the user) is the only attribute used in this study.

The following API methods are utilized for returning a collection of Tweet objects. Every API method accepts various parameters to make a specified query and return a response object.

- *Search API*: The standard search API searches against a sampling of recent Tweet objects published in the past 7 days. It is worth noting that the Search API retrieves only the relevant tweets not an exhaustive source of tweets. The key parameters of Search API include: 1) query, which identifies the search query characters with a maximum of 500 characters, 2) language, which restricts tweets to a given language, and 3) geocode, which returns tweets by users located within a given radius of a given geographic coordinate. This API returns a list of Tweet objects as the response.

- *Get statuses/user_timeline API*: This API method returns a collection of the most recent Tweets posted by the user specified by the screen_name parameter. The screen_name is a unique name that identifies the user profile. This API returns a list of Tweet objects as the response.

- *Get users/search API*: This API runs a search for finding public user accounts on Twitter that are relevant to the search query parameter. The query parameter is typically made by full name, company name, and other keywords. This API returns a list of User objects as the response.

5.3.2 Label definitions

Collected tweets are manually labeled into one of the three following classes:

1. *Non-Traffic (NT)*: Any tweet that does not fall into the other two categories is labeled as NT.

2. *Traffic Incident (TI)*: This type of tweet reports non-recurring events that generate an abnormal increase in traffic demand or reduce transportation infrastructure capacity. The examples of non-recurring events include traffic crashes, disabled vehicles, highway maintenance, work zones, road closure, vehicle fire, traffic signal problems, special events, and abandoned vehicles. Since the ultimate goal of our framework is to inform users and agencies on the occurrence of a traffic incident in a real-time basis, if a tweet reports on the clearance or re-opening of roads that had already been affected by non-recurring traffic events, that tweet is classified as TCI, the third tweet category. Indeed, such tweets are providing information on the current status of the network rather than informing an ongoing traffic incident.

3. *Traffic Conditions and Information (TCI)*: This type of tweet reports traffic flow conditions such as daily rush hours, traffic congestion, traffic delays due to high traffic volume and jammed traffic. Also, any tweets that disseminate new traffic rules, traffic advisory, and any other information on transport infrastructures (e.g., new facilities or changing the direction of a street) are classified as TCI.

5.3.3 Tweet collection and annotation

After describing the API methods for collecting tweets and defining classes for labeling tweets, an efficient methodology is deployed to expedite our manual labeling process. First, a traffic-related dictionary is built by the aid of Twitter's users who mainly disseminate traffic information, also known as Influential Users (IUs). Using the built traffic dictionary and IUs, tweets are collected in three datasets with different procedures. It is worth mentioning that conducting the process of the tweet collection in three different datasets is only for speeding up the labeling task. After manually labeling tweet in each dataset, all tweets are integrated to constitute a unified labeled dataset. In the next sections, the procedure for creating the traffic-related dictionary and collecting tweets in each dataset are described.

5.3.3.1 Traffic-related dictionary The traffic-related dictionary consists of only the most frequent words occurred in traffic-related tweets. A common method in literature for obtaining traffic-related tweets is to gain them from accounts that belong to transportation and emergency service providers (e.g., State Department of Transportation). However, a significant portion of posted tweets by these users are not about events that just happened or is happening in a traffic network. As a consequence, a time-consuming labeling process must be implemented to distinguish traffic-related tweets from non-traffic-related tweets. A more effective way to fetch traffic-related tweets with a very high chance and without the need for manually labeling tweets is to provide a list of Twitter users that primarily post traffic information.

511 is a national traffic information telephone hotline across some regions of the United States that primarily provides traffic-related information for drivers. Several regions have their own 511 account for publishing travel information and traffic alerts. Accordingly, an initial list of Twitter users that are relevant to 511 is created by searching users based on the keywords "511", "Traffic", and "State" through the GET users/search method. "States" is the name of one of the US states. After obtaining the matched users, we simply visualize the recent tweets of the matched users to ensure that these Twitter accounts are real 511 profiles that primarily disseminate traffic incident information. Following this process, a IU list with 69 users such as "511nyNJ, TN511, WV511South, fl511_northeast, and 511northwestva" is obtained. A few examples of traffic-related tweets extracted from IUs are provided in Table 5.2.

Subsequently, a pool of the most recent tweets posted by the users in the IU list are collected using the Get statuses/user_timeline API method. Following the bag-of-words concept, the collected traffic-related tweets are tokenized, and the token-occurrence frequency is computed for each distinct token. Occurrence

Table 5.2: Examples of traffic-related tweets extracted from Influential Users (IUs)

User screen name	**Traffic-related tweets**
511northernva	Cleared: Disabled Vehicle: EB on I-66 at MM68 in Arlington Co.11:36 AM
511Georgia	Accident, I-285 North past Lavista Road, far right lane blocked. #511GA
511NYC	Closure on #ThrogsNeckBridge SB from Bronx side to Queens side

frequency of tokens are then summed up over all tweets. Finally, tokens with the highest occurrence frequency are selected to build the traffic-related dictionary. Since a portion of collected tweets might be non-traffic-related, the selected words are re-investigated one by one to make sure all words have a semantic relationship with traffic and transportation. Almost 60 high-frequent traffic words are used for building the dictionary. Examples of traffic words in the dictionary are "traffic, blocked, lane, construction, crash, congestion, delays, vehicle, incident, ramp, and street".

Having the IU list and traffic-related dictionary in hand, we collect tweets in three datasets with different API methods. As mentioned, the rationale behind collecting tweets with different procedures is to only expedite the labeling process since each dataset brings its own advantages. After manually labeling tweets in these datasets, they are concatenated to form one unified-labeled-tweet dataset.

5.3.3.2 Collecting the first dataset This dataset contains tweets from random users using Search API by specifying the language and geocode parameters to English and the boundaries of the US, respectively. However, returning a traffic-related tweet (i.e., TI and TCI tweets) among all other types of tweets without specifying any search query is similar to outlier detection. Thus, if no keyword is specified for fetching tweets from random users, the chance of receiving traffic-related becomes too low. For addressing this problem and making a balance between traffic-related and non-traffic related tweets, a combination of keywords in the traffic-related dictionary is used for making the search query parameter. Such a query increases the chance of getting traffic-related tweets generated by random users. In each request, a combination of two, three, or four keywords from the built dictionary is used for gathering Tweet objects. Keywords are randomly selected from the dictionary. We aimed to obtain almost 16,000 tweets within the first dataset, which constitutes around one third of the whole dataset. Since our goal is to have the same portions for both traffic-related and non-traffic related tweets from random users, the query parameter is removed after obtaining almost 8,000 traffic-related tweets. Fetching tweets without defining a specific query dramatically soars the chance of getting non-related traffic tweets.

After collecting all tweets for this dataset, the final collected tweets are carefully read by authors and labeled according to the above-mentioned class definitions.

Since tweets in this dataset are fetched from random users, the labeling process takes longer compared to the following two datasets. However, labeled tweets from random users causes more diversity in the final dataset, which in turn improves the generalization capability of the detection model.

5.3.3.3 Collecting the second dataset In this step, we aim to collect only potential traffic-related (i.e., TI and TCI) tweets. Therefore, a collection of tweets posted by IU is retrieved using GET statuses/user_timeline API method. Since tweets posted by IU accounts are most probably about travel information, the labeling process becomes faster. A labeler only needs to look for some keywords in a sentence to decide the appropriate label. Although tweets in this set are potential to be traffic-related tweets, they are manually inspected with a high speed and labeled into one of the NT, TI, or TCI categories. Almost 17,000 tweets from users in the IU list are collected to form the second dataset.

5.3.3.4 Collecting the third dataset In this dataset, we aim to collect potential non-related traffic tweets. Analogous to the second dataset, in order to speed up the process of labeling tweets, tweets from users who almost never post tweets about traffic condition are fetched. A list of 70 Twitter accounts from various areas including fashion industry, politicians, celebrities, irrelevant organizations, etc., are provided. Examples of such users are "AshleyFurniture, AEO, nabp, MikePenceVP, and verizon". Using the API GET statuses/user_timeline method, roughly 17,000 recent tweets from 70 non-traffic-related users are collected. In a similar way to the second dataset, because the chance of retrieving non-traffic-related tweets are high in this dataset, it becomes easier for the labeler to annotate tweets. Tweets are manually inspected, yet with a higher speed, and labeled into one of the NT, TI, or TCI categories.

5.3.3.5 Tweet pre-processing Before labeling the collected tweets in each dataset, the following steps are applied to each tweet:

- Tweets with less than four words, excluding stop words and special characters, are discarded.
- Tweets with the similar text attribute are detected and, except one of them, the remaining ones are discarded. Two tweets are considered similar if they

contain similar words, excluding stop words and special characters and regardless of word positions in the tweet text.

- Remove all web URLs, special characters, punctuation marks, and stop words (e.g., articles and prepositions).
- If the current tweet is the result of re-tweeting another tweet, the original text from the native tweet is used to avoid having a truncated text.
- Make all words lowercase.
- Substitute all U.S. and Interstate Highways with the word "highway". "us number", "i-number", and other similar formats are considered as the patterns for finding U.S. and Interstate Highways in the texts.
- Remove the remaining numbers.

5.3.3.6 Labeling results The label distribution of tweets in the three datasets is shown in Table 3. For having the same number of traffic-related tweets (i.e., TI and TCI) and non-traffic-related tweets (i.e., NT), we trim back some tweets. The last row in Table 5.3 shows the tweet distribution among three labels as the final labeled-tweet dataset after removing some tweets. We release the whole dataset as a possible benchmark for future researchers in this type of application. To the best of our knowledge, we are the first research group who release the largest labeled-tweet dataset in a traffic domain.

In this study, we build two types of datasets from the unified labeled-tweet dataset: 1) 2-class dataset, in which tweets are categorized into traffic-related tweets (i.e., TI and TCI) and non-related-traffic tweets (i.e., only NT). 2) 3-class dataset, in which tweets are categorized into three groups including NT, TI, and TCI.

5.4 Methodology

CNNs and RNNs are two widely used deep learning architectures that have achieved great success in the text classification problem [45, 50, 82, 101]. Accordingly, these two supervised algorithms are deployed for detecting traffic-related tweets. Among

Table 5.3: Number of tweets in each dataset and class

		Traffic-Related		
Dataset\Class	**NT**	**TI**	**TCI**	**Total**
First	8,138	5,414	2,651	16,203
Second	377	12,022	5,462	17,861
Third	18,259	2	0	18,261
Total	26,774	17,438	8,113	52,325
Unified dataset	25,550	17,437	8,113	51,100

several variants of RNNs, the standard Long Short-Term Memory (LSTM) is utilized. Figure 2 depicts three architectures for classifying tweets: 1) only CNN, 2) only LSTM, 3) combination of CNN and LSTM. Inspiring by Kim [45], these networks are trained on the top of tweets' word vectors, which are obtained through well-known word-embedding tools rather than the traditional bag-of-words. In the following sections, the mechanism of word embedding, CNN, LSTM, and the way they interact with each other for identifying tweets' labels are elaborated.

5.4.1 Word embeddings

Word-embedding tools map millions of words to vectors of real numbers in such a way that words with similar meanings tend to be closer to each other in vector space. In addition to capturing the semantic similarity, word-embeddings provides a low-dimensional vector with tens or hundreds of dimensions for each word. This is in contrast with the one-hot encoding method where every word is represented by a very sparse and high-dimensional vector. Dense representations improve the generalization and computational power of learning models including neural network toolkits [32]. To date, several word-embedding models have been proposed to learn a distributed representation for every word in a text corpus. The resulting feature vectors can be utilized in a variety of challenging NLP tasks such as text classification. In this paper, we deploy two sophisticated and well-known word-embedding tools for tweet modeling: word2vec and FastText models.

5.4.1.1 Word2vec model Word2vec model is an unsupervised learning algorithm that deploys a Neural Network Language model (NNLM) to learn high-quality distributed representations of words in vector space from a massive data sets with millions of words in the vocabulary [55]. Skip-gram and Continuous Bag-of-Words (CBOW) are two main architectures of the word2vec model, where a neural network with one hidden layer is trained for each type. In the skip-gram architecture, the current word is used as an input and the main task is to predict the occurrence probability of every word in the vocabulary to be within a certain range before and after the current word. In other words, the output probabilities indicate how likely each vocabulary word is in the context of the input word. The input layer receives a one-hot vector corresponding to the input word while the output layer produces probability distribution for all words in the vocabulary. Thus, the number of neurons in the input and output layers are equal to the number of words in the vocabulary, denoted as v. The hidden layer size is set to the dimensionality of the word vectors, denoted as d.

Thus, the weight matrix $\mathbf{W}$ between the input and hidden layer has the size (v). Given a sequence of training words $[w_1, w_2, w_3, ..., w_T]$, the objective of Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{K}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{5.1}$$

where c is the context size with typical values of 5 or 10. After training the model, the main output is the weight matrix $\mathbf{W}$, where each row in $\mathbf{W}$ corresponds to the vector representation of a word in the vocabulary. Intuitively, since words in a similar context need to generate almost similar outputs, the model is forced to learn similar feature vectors for words appeared around each other in the training corpus. CBOW, another architecture for word2vec models, is similar to the skip-gram, yet predicts the current word based on its context window. Details for efficiently training the word2vec models are provided in [55].

5.4.1.2 FastText model Word2vec models assign a distinct feature vector to every word of the vocabulary while ignoring the internal structure of words. FastText is an extension of the word2vec model where each word is represented as a bag

of character n-grams [8]. Thus, the Skip-gram model is designed to learn a vector representation for each character n-gram in the text corpus. Then, the vector representation for a word is the sum of its associated n-gram vectors. Taking the sub-word information into account allows to have a representation for rare words that have not been appeared in the training corpus. In particular, FastText is capable of improving word representations for morphologically rich languages that contain verbs and nouns with a variety forms, which might rarely occur in the training corpus.

5.4.1.3 Creating embedding layer Using the word-embedding tools, every tweet is mapped into a fixed-sized feature matrix, called embedding matrix. Word-embedding matrixes of tweets in each training batch need to have the same size for both RNN and CNN algorithms. Thus, a maximum sequence length, denoted as L, should be defined for all tweets that have variable number of words. An appropriate approach to define the maximum length is to observe the percentile values associated with the distribution of number of words in tweets. This approach will be discussed in the next section. Tweets with the number of words more than L are truncated to L while shorter tweets are padded with zero values.

For creating the embedding matrix of a tweet, first the tweet is tokenized into a group of words (i.e., unigrams). As an example, pre-processing and tokenizing the first tweet in Table 5.2 results in: ['cleared', 'disabled', 'vehicle', 'eb', 'highway', 'MM68', 'arlington', 'am']. Next, the word vector of each token is retrieved from one of the word-embedding models. Concatenating word vectors of up to L tokens generates the word-embedding matrix $\mathbf{X}$ with size $(L \times d)$ for every tweet, where d is the word-vector dimension. The matrix $\mathbf{X}$ is regarded as the tweet representation. Therefore, unlike the previous methods in literature, no predefined set of traffic-related keywords is used to model the tweets in numerical feature vectors. Moreover, unlike the bag-of-words representation, the word-embedding matrix not only capture the semantic and syntactic relationship between tweets words but also does not suffer from the curse of dimensionality since d is much lower than the vocabulary size of the tweet corpus. Finally, a group of word-embedding matrixes is fed into the CNN or LSTM layer as the input layer for all architectures shown in Figure 5.2.

5.4.2 Convolutional Neural Networks

Although the key idea in CNNs is similar to ordinary feed-forward artificial neural networks, they differ in terms of connectivity patterns between neurons in adjacent layers. The CNN takes the advantage of spatially local correlation by connecting neurons to only a small region of the preceding layer, also called receptive field. In the convolutional layer, this is achieved by convolving a filter across the whole surface of the previous layer output volume. The output of convolution operation for each filter is called a feature map. Concatenating feature maps of multiple filters results in a new volume. Such capability of CNNs in learning local response enables CNNs to capture abstract representation of n-grams through convolutional filters. This leads CNNs enjoying a great success in natural language applications including text categorization.

The CNN model in this study is a slight variant of the model proposed in [45]. In the convolutional layer of our architecture, a filter $\mathbf{F}$ has size $(n \times d)$, where the n is the number of consecutive words (i.e., n-grams) in a tweet and d is the width of filter equal to the word-vector dimension. It is very important to choose the width of the filter equal to the word-vector dimension since the principal goal is to extract high-level representation of n-gram features. Sliding the filter $\mathbf{F}$ across the matrix $\mathbf{X}$ produces a feature map $\mathbf{c}$ with size $(L - n + 1)$ as follows:

$$\mathbf{c} = \left[c_1, \cdots, c_i, \cdots, c_{L-n+1}\right]^T \tag{5.2}$$

where c_i is computed according to Eq. 5.3:

$$c_i = f(\mathbf{X} \circ \mathbf{X}_{i:i+n-1} + b) \tag{5.3}$$

where $\circ$ is the dot product between the parameters of filter $\mathbf{F}$ and the entries of submatrix $\mathbf{X}_{i:i+n-1}$, b is a bias term, and f is a non-linear activation function. Rectified Linear Units (ReLU) is utilized as the activation function throughout this paper. Using K filters of the same size as $\mathbf{X}$, K feature maps are generated. The filters and their corresponding feature maps have been color-coded in Figure 5.2. Horizontally concatenation of the created feature maps results in a new word-vector representation matrix $\mathbf{X}_{new} : [L - n + 1, K]$. The i-th row of the matrix $\mathbf{X}_{new}$ is a new feature representation of the n-grams in the original matrix $\mathbf{X}$.

For the CNN-based architecture, as shown in Fig. 5.2a, a max-pooling layer with filter size $(L - n + 1, 1)$ is applied to $\mathbf{X}_{new}$. The rationale behind applying the max-pooling layer is to take one feature with the highest value (probably as the most important one) from each feature map c. Therefore, the final vector that is fed into the softmax layer for the classification task is a vector with size $(1, K)$, where each element is corresponding to one feature map.

Nonetheless, for the CNN+LSTM architecture in Fig. 5.2c, since the output of the CNN layer is fed into the LSTM layer, the max-pooling layer is not used after the convolutional layer. As will be described in the next section, every time step in LSTM represents the vector of a word (i.e., unigrams) or the vector of a number of consecutive words (i.e., n-grams). Applying the max-pooling layer generates a sequence where each element is a kind of text representation not word(s) representation. Thus, $\mathbf{X}_{new}$ is directly passed into LSTM layer without using the max-pooling layer in the integrated CNN and LSTM architecture.

5.4.3 Recurrent Neural Networks

An RNN is a chain-like neural network architecture that allows information to be passed from one step to another step. Thus, they constitute a series of repeating modules of neural networks, where each module consists of a hidden state h and optional output y that operates on sequential data like $\mathbf{X} = \left(x_1, ..., x_t, ..., x_T\right)$ [13]. Hidden state h can be imagined as a hidden layer inside each module with a specific number of units, denoted as S, which is an important hyperparameter in RNN models. Each module looks at the current input x_t and the previous hidden state h_{t-1} to update the hidden state h_t as follows:

$$h_t = f(x_t, h_{t-1}) \tag{5.4}$$

where f is a non-linear activation function. Various versions of RNNs differ in the structure of the function f. In the vanilla RNN, f has a very simple structure such as an element-wise *sigmoid* or *tanh* layer. However, the standard RNN is unable to learn the long-term dependencies, which occurs when the gap between relevant information at one timestamp and the needed timestamp becomes large. The rationale behind such incapability is that, during the backpropagation phase,

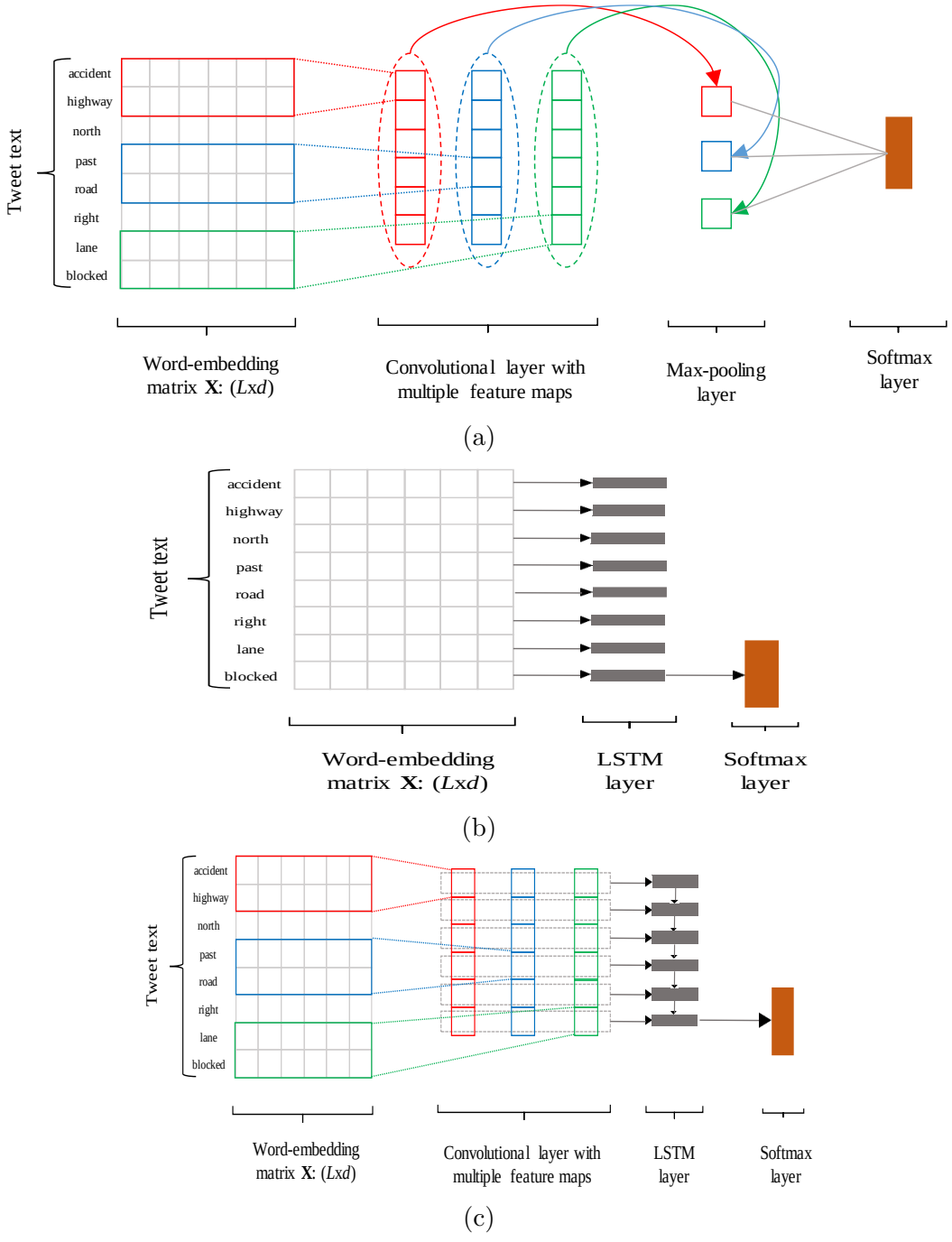


Figure 5.2: Deep-learning architectures for tweet classification. (a) only CNN, (b) only LSTM, (c) integrated CNN and RNN. Each color code in the convolutional layer is corresponding to one filter and its feature map. In this example, the height of filters is 2 (i.e., they convolve bigrams).

the gradients are becoming either too small or too large, also referred as vanishing and exploding gradients, respectively. These issues corrupt the process of learning.

To address such drawbacks, the LSTM network was introduced by Hochreiter and Schmidhuber [38] to capture long-term dependencies and avoid the vanishing/exploding gradient problems. Indeed, LSTM inherits the main properties of standard RNNs yet with more controls on how to pass the information in the repeating modules. The key element in each LSTM module is its cell state C_t that needs to be updated using two gates: forget gate f_t and input gate i_t. The forget gate and input gate decide to what extent the information from the previous cell state as well as the new candidate cell state $\hat{C}_t$ are needed to be kept. Then, the new cell state values are updated by combing these two new vectors as follows:

$$\begin{aligned} f_t &= \sigma\big(W_f[x_t, h_{t-1}] + b_f\big) \\ i_t &= \sigma\big(W_i[x_t, h_{t-1}] + b_i\big) \\ \hat{C}_t &= tanh\big(W_c[x_t, h_{t-1}] + b_c\big) \\ C_t &= f_t \circ C_{t-1} + i_t \circ \hat{C}_t \end{aligned} \tag{5.5}$$

where σ and $tanh$ are the logistic sigmoid and hyperbolic tangent functions, respectively. W_f, W_i, W_c, b_f, b_i, and b_c are the trainable weight matrices and biases. Finally, the LSTM module needs to decide on its hidden state as the output. The hidden state at each time indicates what parts of the cell state is going to be output, which is defined by the output gate o_t as follows:

$$\begin{aligned} o_t &= \sigma\big(W_o[x_t, h_{t-1}] + b_o\big) \\ h_t &= o_t \circ \tanh C_t \end{aligned} \tag{5.6}$$

where W_o and b_o are the weight matrix and bias, respectively, for the output gate layer. Notwithstanding that many variants of LSTMs exist in literature, in this project, the standard LSTM as described above is used.

In the only LSTM architecture in Figure 5.2b, the input layer created based on the word-embedding matrix $\mathbf{X}$ is fed into the LSTM layer. Thus, the current input at the step t, x_t, is the word vector corresponding to the t-th word in the tweet.

However, for the CNN+LSTM architecture in Figure 5.2c, the input layer for the LSTM layer is the matrix $\mathbf{X}_{new}$ as obtained through the CNN layer. Finally, for both only LSTM and CNN+LSTM architectures, the hidden state in the last step of the LSTM layer is considered as the vector of tweet representation and is fed into the $softmax$ layer. The $softmax$ function in the last layer performs the classification task by generating a probability distribution over tweet labels. As mentioned, both LSTM and CNN+LSTM architectures in this study have been developed based on the proposed model in [101].

5.4.4 Training process

The described architectures are trained by optimizing the categorical cross-entropy as the loss function. The Adam optimizer is used to update model parameters in the back-propagation process. Adam is a well-suited optimization technique for problems with large dataset and parameters that has recently seen broader adoption for deep learning applications [46]. We use the batch size equal to 64 and Adam's default settings as provided in their paper: $learning_rate = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The network weights are initialized by following the proposed scheme in [30].

In terms of regularization, the dropout method is utilized as the most practical and widely used approach in deep learning architectures [78]. For all architectures, the dropout layer is added before the last layer (i.e., the softmax layer) with setting the dropout ratio to 0.5. Furthermore, the early-stopping method is implemented, which is another approach to avoid overfitting. The training is stopped if the performance metric does not improve on the validation set after two consecutive epochs. The model with the highest validation score is restored for applying on the test set.

5.5 Experimental results

In this section, the performance of the proposed model for detecting traffic events from Twitter streams is evaluated on the labeled tweets, described in the previous Section. First, the baseline methods for comparison against the proposed model

and the performance metrics for evaluation are introduced. After tuning the models' hyperparameters, several rounds of quantitative and qualitative evaluations are conducted to assess the proposed framework performance in various situations.

5.5.1 Experimental setup

5.5.1.1 Baseline methods Our model performance is compared against several baselines. The most relevant studies, which have leveraged Twitter data to detect traffic-related events, have been summarized in Table 5.1. Those works that heuristically classified tweets based on syntactic analysis rather than numerical feature vectors are discarded (i.e., the first three studies in Table 5.1). In the following paragraphs, the proposed techniques in each study are explained at the high-level and details are left to their corresponding references. It should be noted that all studies are almost similar in terms of the classification task, yet different in tweet modeling. Hence, we focus more on how they numerically represent tweets before feeding them to a machine learning classifier.

- *Carvalho 2010*: First, stop words and punctuation marks were removed. Then, they used linear SVM for classifying tweets that are represented through the bag-of-words method while considering either uni-grams or bi-grams as features. Since this model contains less details compared to the following models for tweet representation, we classify tweets using two more well-known supervised learning algorithms including Random Forest (RF) and Multilayer Perceptron (MLP). This gives room to observe the performance of other classic machine learning algorithms in addition to SVM.

- *D'Andrea et al. 2015*: After applying some preprocessing steps including stop-word filtering and word stemming, the tweet corpus was turned into numerical feature vectors using the bag-of-words representation. idf was chosen as the weight value in the bag-of-words matrix. Next, the most relevant features were selected by computing Information Gain (IG) for all features. Features with positive IG were selected and fed into the SVM algorithm for the classification task.

- *Fu et al. 2015*: First, traffic-incident-related words with the highest tf-

idf weights were extracted from tweets posted by four IUs accounts. Using the Apriori algorithm, the most frequent word set were identified to build more efficient queries for crawling traffic-related tweets through Twitter APIs. Finally, extracted tweets were scored and ranked based on the sum of traffic keywords tf-idf weights. However, it is not known how tweets were classified. One way is to label the extracted tweets using the most frequent wordset and label them as traffic-incident tweets while leaving others as non-traffic tweets. The other potential approach is to label a tweet as a traffic-incident one if its score exceeds a certain threshold (e.g., 50 percentile of all scores) while labeling others as non-traffic incidents. In general, the process of labeling tweets is very vague in this paper. However, we report the best accuracy that can be achieved out of their method.

- *Gu et al. 2016*: The core part of their procedure was the adaptive data acquisition, which establishes a dictionary of words that are positively correlated with traffic incidents. Following the standard bag-of-words representation, all tweets were projected onto this feature space and classified into traffic incident and non-traffic incident categories using the Semi-Naïve-Bayes classifier. In the adaptive data acquisition, they first created an initial set of words related to traffic incidents such as traffic, accident, etc. At each iteration, the dictionary was first expanded by adding two synonyms of every word using the open-source WordNet database. Then, queries with one word or a combination of two, three, and four words from the dictionary were constructed to crawl tweets through Twitter APIs. Next, tweets were manually labeled. After tokenizing tweets, a "reducer" was applied to all acquired and labeled tweets to count the total number of positive and negative labels for all token combinations. Token combinations with the maximum positive and negative counts were selected and added to the dictionary for the next round of acquiring tweets. The algorithm goes on until the number of retrieved traffic-incident tweets becomes lower than a specified threshold. Positively correlated words with traffic in the built dictionary were served as features for the classification task.

- *Pereira et al. 2017*: Two groups of features were utilized for creating feature vector space: 1) standard bag-of-words, 2) word embeddings. These two

pools of features were then horizontally concatenated to represent the feature space for the classification task. Using the created feature vectors, classical supervised learning algorithms (e.g., SVM) were deployed to identify travel-related tweets from non-related ones. The same as in our proposed model, we utilized either of Google or Twitter word2vec models to generate bag-of-word embeddings. This also leads to have a fair comparison.

Unfortunately, neither the codes nor the tweet data are available for these studies. Accordingly, we have done our best to reproduce their frameworks exactly the same as what authors have described in their papers. During the implementation, we assume a reasonable setting for any case that the required information has not been provided. Furthermore, their models are trained and tested on our own datasets to have a fair comparison. Performance metrics

The performance of our proposed model and baselines are evaluated using two common classification metrics:

- *Accuracy* - it is computed as the fraction of tweets in the test set that are correctly classified.
- *F-measure* - F-measure for every tweet class is defined as the harmonic average of its precision and recall in the test set, as shown below:

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

The final $F - measure$ for a model is the unweighted average of $F - measure$ for each class. Unweighted F-measure is a more reliable metric for problems with imbalanced classes, which is the case in the 3-class dataset and the real-world situation.

In all our 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-fold are reported. In other words, we create 5 train/test splits from the whole dataset. In each split, 4-fold is used for training while holding out the remaining one-fold as the test set. Before obtaining the average results on the 5 test folds, models' hyperparameters are tuned. To this end, one-fold out of the 4-fold training

is used as the validation set for each split. The hyperparameter combination that on average achieves the highest performance on the validation sets of all splits is used for the final training and testing on 5 train/test splits. In this case, the test fold in each split plays no role in tuning hyperparameters. Next, using the obtained hyperparameters, models are trained on the 4-fold and tested on 1-fold for each train/test split and the average results are reported. By doing this, every tweet in the whole dataset is used in the test set at least and at most once. Only for implementing the early-stop method in deep-learning models, 10% of the training data in each split is randomly selected as the validation set for the early-stopping procedure using the stratified random selection.

All data processing and models are implemented within Python programming language. Tweepy, a python wrapper, is used for access to the entire APIs methods. Collected Tweet objects are stored in MongoDB as a non-relational database since the number of attributes vary among Tweet objects. The deep learning architectures are learned in [14] using the TensorFlow backend with the GPU support. Classic machine learning algorithms are implemented using the scikit-learn library. The labeled data and source codes related to all data processing and models utilized in this study are available at https://github.com/sinadabiri/Tweet-Classification-Deep-Learning-Traffic.

5.5.1.2 Word-embedding matrix Word vectors can be considered as non-static parameters and simultaneously trained with other trainable weights. However, pre-trained word vectors trained on a large corpus often bring about high-quality performance in NLP tasks [45, 101]. Thus, in all our experiments, the word-embedding matrix of each tweet is initialized using publicly available word vectors. Two sets of pre-trained word vectors are utilized: 1) word vectors trained on 400 million tweets using the word2vec model. Its weight matrix W contains 400-dimensional vectors for almost 3 million words and phrases [31]. 2) word vectors trained on Wikipedia data using the FastText model. Its weight matrix **W** contains 400-dimensional vectors for almost 2.5 million words [8].

Our analysis reveals that the average number of words not presenting in the pre-trained word2vec and FastText word vectors are 0.9 and 1.0 words per tweet, respectively. Accordingly, since both word-embedding models contain a great portion of words in a tweet (i.e., missing almost one word per tweet), the word-

(a) (b)

Figure 5.3: Distribution of number of words presented in word-embedding models per tweet. (a) word2vec, (b) FastText

embedding matrix is built based on only words presented in pre-trained word vectors. This avoids having a word vector that has not been pre-trained through word-embedding tools.

For each word-embedding model, the maximum length of a tweet is selected according to the distribution of the number of words per tweet, which have been shown in Figure 3-a and 3-b for word2vec and FastText models, respectively. The value corresponding to a high percentile can be appropriate as it includes more words for lengthy tweets. With regard to Figure 5.3, 90 percentage of tweets have less than 13 words for both word2vec and FastText word vectors, respectively. Thus, the maximum length parameter L is set to 13 when using either of pre-trained word vectors.

5.5.1.3 Hyperparameters Since a deep learning model contains a large number of hyperparameters, a practical approach for defining appropriate values of hyperparameters is to either observe their effect through conducting several small-scale experiments or leaving them with their default values used in the most seminal studies. Hyperparameters relevant to the training process (e.g., drop out ratio), reported in Section 5.4.4, have been set in this way. However, the important hyperparameters in our models, which are the filter size n and the number of filters K in the CNN architecture as well as the number of units S in the LSTM architecture, can be selected using the grid search based on the average metric. Table 5.4 provides the average accuracy results for every pair of (n, K), where n and K vary in the range of $n \in [1, 5]$ and $K \in [5, 200]$. As can be seen in Table 5.4, increasing K more than 200 does not improve the prediction quality. Also, convolving bi-grams (i.e., $n = 2$) engenders the highest accuracy for a majority of K values. Therefore, the combination of $n = 1$ and $K = 200$ is selected for the CNN architecture. Table 5.5 represents the one grid search for S, which varies in the range $n \in [5, 200]$. From Table 5.5, S is set to 30 since it is the minimum number of LSTM units that results in the highest accuracy.

Table 5.4: Average accuracy results for various combinations of n and K in the CNN architecture

		Number of filters									
		5	15	25	40	60	80	100	150	**200**	250
Filter Size	**1**	0.966	0.978	0.981	0.981	0.982	0.983	0.984	0.984	0.985	0.984
	2	0.975	0.979	0.982	0.982	0.983	0.984	0.984	0.984	0.985	0.984
	3	0.974	0.981	0.981	0.983	0.983	0.983	0.983	0.984	0.984	0.984
	4	0.975	0.980	0.981	0.982	0.982	0.982	0.982	0.983	0.982	0.983
	5	0.972	0.980	0.981	0.982	0.982	0.983	0.983	0.983	0.983	0.982

Table 5.5: Average accuracy results for various values of S in the LSTM architecture

Number of units								
5	10	20	**30**	40	50	100	150	200
0.982	0.982	0.982	**0.983**	0.983	0.983	0.982	0.981	0.981

Note that the grid search has been applied to the 2-class dataset and the pretrained word2vec vectors, yet they will be used for evaluating the models' performance on the 3-class dataset and FastText vectors as well. The number of layers in each architecture is another principal hyperparameter in such architectures. However, since increasing the models' depth does not improve the performance quality, the number of CNN and/or LSTM layers is set to 1 for all three architectures to keep the model simple and reduce the computation time.

5.5.2 Results and discussion

In this section, we first assess the performance of the proposed architectures on its own by reporting and discussing the evaluation metrics. The proposed deep-learning models are then evaluated by comparing with baselines and simulating a real-world situation. The learning behavior and the confusion matrix of associated with the proposed model are also examined. Finally, a qualitative assessment is conducted to investigate for what types of tweets the proposed model is unable to correctly predict labels.

5.5.2.1 Model performance Before proceeding to the comparison results, various combinations of word-embedding models and neural-network classifiers are internally evaluated. Table 5.6 shows the accuracy and F-measure of three architectures trained on the top of word2vec and FastText models for classifying tweets into two or three labels. As can be seen from Table 5.6, all three types of deep learning models perform nearly the same as each other for the 2-class and 3-class datasets in terms of both accuracy and F-measure. However, CNN on the top of word2vec outperforms other models by almost 0.2% and achieves the state-of-the-art accuracy and F-measure of 0.986 for the 2-class dataset. In the meantime, the proposed models maintain their quality prediction while the number of labels increases to the 3 classes. The performance measures reduce a bit from 0.986 to 0.974 for the 3-class datasets. The high value for both performance metrics at the same time is another evidence that the deep-learning models perform well. Since the CNN architecture with the word2vec model has obtained the best result, this model is used as the representative of deep-learning models for the remaining analyses in this Section.

Furthermore, the proposed architectures are trained based on randomly initialized word vectors so as to evaluate the efficacy of word-embedding tools. Matrix X is randomly initialized from a uniform distribution between the maximum and minimum values of all vectors in the pretrained word2vec model. Closer inspection of Table 5.6 reveals that the models obtain a high rate of accuracy when a well-trained word-embedding model is used for tweet representation. Although pretrained word2vec and FastText models have trained on distinct datasets with different sizes and domains (i.e., tweet and Wikipedia corpora), the deep-learning networks attain similar prediction results on both sorts of word-embedding models. Comparing the accuracy and F-measure obtained through word embedding models with randomized word vectors demonstrates the effectiveness of word-embedding tools in modeling tweet words. Using the same CNN and LSTM configurations yet with randomized vector representation of words dramatically decreases the accuracy and F-measure to almost 50% for both 2-class and 3-class datasets.

5.5.2.2 Comparison results Table 5.7 summarizes the performance metrics of the CNN architecture and baselines for both the 2-class and 3-class problems. Every value indicates the average results in 5-fold cross-validation along with their

Table 5.6: Accuracy and F-measure of three deep learning architectures, which have been trained on word2vec, FastText, and random word vectors. Acc and F1 correspond to accuracy and F-measure, respectively.

	word2vec				**FastText**				**Random**			
	2-Class		**3-Class**		**2-Class**		**3-Class**		**2-Class**		**3-Class**	
Model	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
Only CNN	**0.986**	**0.986**	**0.974**	**0.974**	0.986	0.986	0.973	0.973	0.501	0.398	0.493	0.370
Only LSTM	0.984	0.984	0.972	0.972	0.985	0.985	0.970	0.970	0.503	0.502	0.500	0.336
CNN+LSTM	0.985	0.985	0.972	0.972	0.986	0.986	0.971	0.971	0.499	0.498	0.500	0.334

corresponding standard deviation. The results of the *Fu* model for the 3-class problem are not reported since their model can be only applied to the binary classification. As can be seen in Table 5.7, the CNN model consistently outperforms the baselines methods in terms of both accuracy and F-measure. The accuracy and F-measure of the deep-learning model beats the best baseline (i.e., the *Pereira* model) with more than 2% for both 2-class and 3-class datasets. A possible explanation for the superiority of the Pereira model compared to other studies might be using word-embedding features in addition to bag-of-words features. For an overall comparison, the CNN model achieves on average 3.5% and 4.9% higher accuracy compared to all baselines for 2-class and 3-class problems, respectively, excluding the *Fu* model which has by far the lowest quality. Except the *Fu* model, other models that have only been trained upon the bag-of-words concept obtain almost the same accuracy. Implementing variants of the *Carvalho* model indicates supervised algorithms including RF and MLP perform either similar to or worse than SVM. In summary, compared to the methodologies in literature, deep-learning architectures not only performs better than baselines but also overcome shortcomings related to tweet modeling.

5.5.2.3 Evaluation in a real-world situation The performance of the deep-learning model has been examined following the common stages for evaluating supervised classifiers. However, our evaluation has not yet taken the real-world situation into account. The ultimate goal of the Twitter-based traffic information system is to detect traffic events from a large volume of Twitter streams, in which only a very small portion of tweets might be relevant to traffic events. Furthermore, since such a system is supposed to work on a real-time basis, the model needs

Table 5.7: Performance comparison between deep-learning methods and relevant studies in literature. NA: Not Applicable

	2-Class		3-Class	
Model	**Accuracy**	**F-measure**	**Accuracy**	**F-measure**
Carvalho-SVM (2010)	0.963(±0.001)	0.963(±0.001)	0.951(±0.001)	0.951(±0.001)
Carvalho-RF (2010)	0.952(±0.002)	0.952(±0.002)	0.916(±0.006)	0.915(±0.007)
Carvalh-MLP (2010)	0.963(±0.001)	0.963(±0.001)	0.946(±0.001)	0.946(±0.001)
D'Andrea et al. (2015)	0.929(±0.003)	0.929(±0.003)	0.882(±0.005)	0.883(±0.004)
Fu et al. (2015)	0.653(±0.002)	0.469(±0.004)	NA	NA
Gu et al. (2016)	0.951(±0.001)	0.951(±0.001)	0.933(±0.001)	0.933(±0.001)
Pereira et al. (2017)	0.965(±0.001)	0.965(±0.001)	0.952(±0.001)	0.952(±0.001)
Only CNN	**0.986(±0.001)**	**0.986(±0.001)**	**0.974(±0.001)**	**0.974(±0.001)**

Table 5.8: Average accuracy and F-measure results of the CNN model for a real-world situation with various amounts of traffic-related tweets. Acc and F1 correspond to accuracy and F-measure, respectively.

	Various proportions of traffic-related tweets in the test set											
	0%		**2%**		**4%**		**6%**		**8%**		**10%**	
Dataset	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
2-class	0.981	0.990	0.980	0.983	0.981	0.982	0.983	0.983	0.985	0.985	0.981	0.981
3-class	0.984	0.992	0.984	0.986	0.982	0.983	0.985	0.985	0.982	0.983	0.981	0.982

to be first trained on the past tweets and then examined on the future data. In order to simulate the real-world scenario, all labeled tweets are sorted based on their creation-time attribute. The first half of the sorted tweets is selected as the training set while the remaining is hold out as the test set. The trained model is evaluated on the test set with all non-traffic tweets plus various proportions of traffic-related tweets. Table 5.8 shows the average accuracy and F-measure results of the CNN model for test sets with different amounts of traffic-related data. For each proportion, five test sets are created by randomly selected traffic-related tweets, and then the average results are reported. The proportions of traffic-related data vary from 0 to 10 percentages to reflect the real-world situation with a small number of traffic-related tweets. It is apparent from Table 5.8 that the CNN model preserves its high-quality performance the same as before and for all amounts of traffic-related tweets.

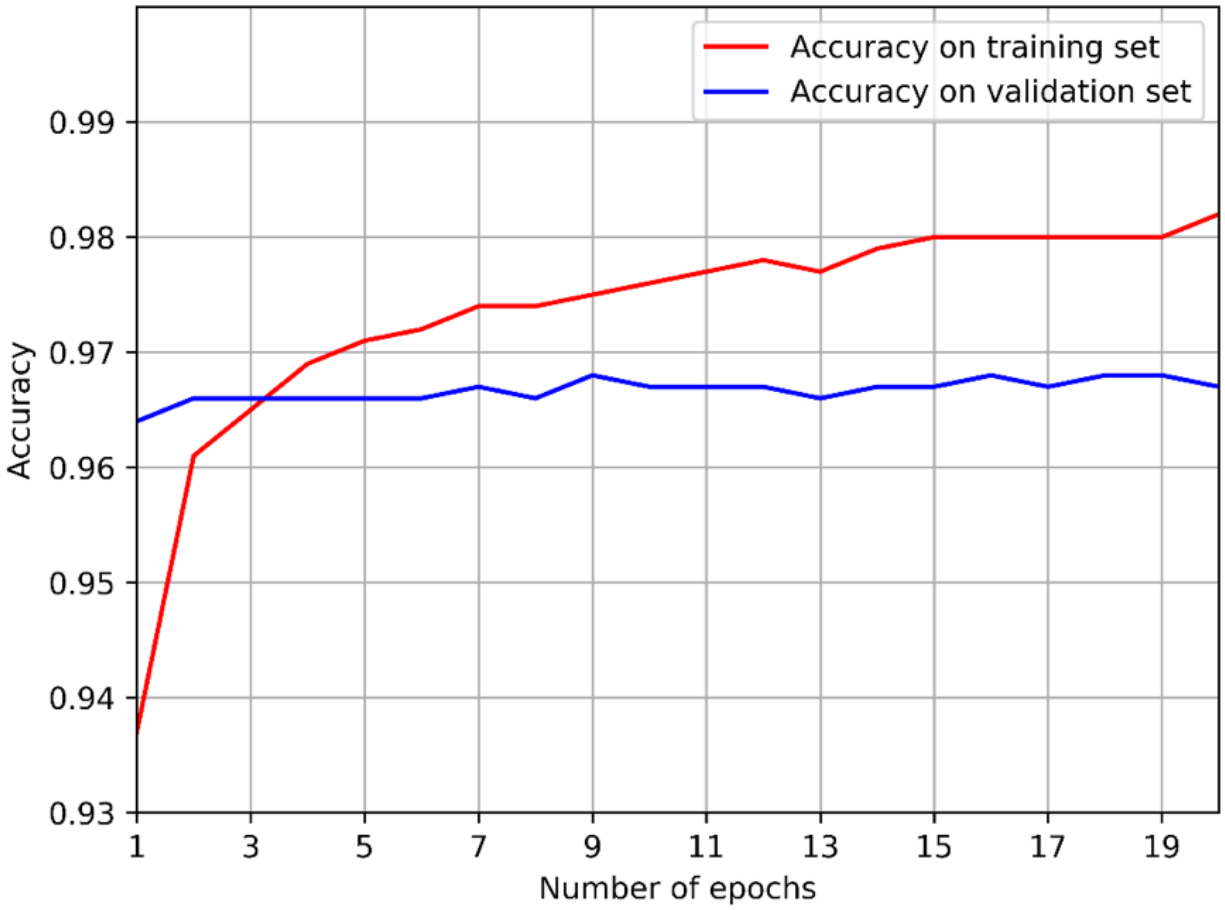


Figure 5.4: Monitoring the performance of the deep learning model by comparing the test and training accuracy for varying number of epochs.

5.5.2.4 Monitoring the model behavior Another interesting analysis is to monitor the performance of the deep-learning models by varying the number of epochs. Figure 5.4 illustrates the accuracy of the CNN model on training and validation sets of the 3-class dataset for various number of epochs. Similar behavior is observed for LSTM and CNN+LSTM models. Such an observation gets insight into diagnosing the potential overfitting or underfitting problem. As can be seen in Figure 5.4, the validation accuracy converges very fast and remains stable after a couple of epochs. Also, since the gap between the training and validation accuracy is very low (i.e., less than 2%) and the validation accuracy does not significantly drop with increasing the number of epochs, we can ensure the overfitting problem does not occur in the proposed models. Note that a high number of training epochs

shown in Figure 5.4 are only used for the purpose of examining the model behavior. However, the training will not continue over all epochs as the early-stopping method stops the training after two consecutive epochs without any improvement.

Furthermore, the average training times are approximately 1.5, 3.0, and 4.0 minutes for CNN, LSTM, and CNN+LSTM models, respectively, which demonstrates the model scalability. Using only one CNN or LSTM layer, training over a few epochs, and short texts of tweets are the primary reasons for the low computation time.

5.5.2.5 Confusion Matrix It would also be interesting to investigate the prediction quality of the proposed model for each tweet class, using the confusion matrix. Table 5.9 and 5.10 report the confusion matrix, precision, and recall for a test set of 2-class and 3-class problems, respectively. In general, the high precision and recall for all classes prove the effectiveness of the model for detecting tweets in each tweet category. As expected, the performance metrics for the TCI class is lower than the other two due to lower number of TCI tweets. Another interesting yet reasonable finding with regard to the confusion matrix in Table 5.10 is that the trained classifier has more difficulty in distinguishing between TI and TCI tweets, in which a majority of false negative tweets for TI belongs to TCI (and vice versa) rather than NT.

Table 5.9: Confusion matrix, recall, and Precision for CNN-based model in 2-class dataset

CNN-based for 2-class dataset		Predicted Class		
		NT	TI&TIC	Recall %
Actual Class	NT	5,031	57	98.3
	TI&TIC	45	3,411	97.6
	Precision %	98.5	96.3	

5.5.2.6 Qualitative Assessment Since the proposed models have achieved a high-quality performance, we are more interested to comprehend for what type of tweets the trained networks are unable to predict the correct label. Table 5.11 presents one example for every combination of miss-classfication in the test set. Their true label and the predicted probability distribution over tweet classes have also been

Table 5.10: Confusion matrix, recall, and precision for CNN-based model* in 3-class dataset

CNN-based for 3-class dataset		Predicted Class			
		NT	TI	TIC	Recall %
Actual Class	NT	5,031	57	32	98.3
	TI	45	3,411	40	97.6
	TIC	34	73	1,513	93.4
	Precision %	98.5	96.3	95.5	

provided. Note that the label with the maximum probability is the predicted label. As can be seen in all examples, there are some ambiguities in the tweets that might even mislead humans. For example, the first tweet in Table 5.11 is indeed reporting an incident with its street-level address yet it is not related to traffic. The tweet #4 belongs to the TI group since a non-recurring reason (i.e., road constriction) caused the delay. However, the classifier has only recognized the existence of delay in a route. On the other hand, the last tweet is describing traffic flow conditions by words "jammed" and "moving 80mph"; however, the classifier might consider it as TI due to using "weird fog light".

Table 5.11: Examples misclassified tweets and their prediction probability over three classes

#	**Tweet texts**	**Prediction Probability**			**Actual Class**
		NT	**TI**	**TCI**	
1	Structural Incident in East Harlem: Due to an unstable building on 108th St, emergency personnel are in the area ...	0.1	99.9	0.0	NT
2	Carneros highway junction could have a (relatively) cheap congestion fix. #Napa #Traffic #Travel	0.5	0.1	99.4	NT
3	UPDATE: Demonstration is now traveling south on 5th Ave from Blanchard St.	75.0	17.7	7.3	TI
4	Penn State football fans can expect traffic delays due to ongoing road construction on U.S. Route.	0.8	39.5	59.7	TI
5	Creek County: SH48 at 41st Street South is back open at this time. Trooper is still working on the scene, use caution	81.2	15.5	3.3	TCI
6	Weird light fog between Cheyenne and chugwater, wy Traffic still moving 80mph Rest area and gas station jammed	0.3	61.4	38.3	TCI

5.6 Conclusion and future work

In this article, we proposed a Twitter-based traffic-event detection framework to be used as a complementary source for monitoring traffic conditions and detecting traffic incidents. First, a large volume of tweets was collected through Twitter API endpoints and then labeled through an efficient strategy. Some effective shortcuts were also implemented in our labeling process to obtain results fast while preserving the quality of labeled data. Using the labeled data, a deep-learning model was proposed for detecting traffic-related events from Twitter streams. First, tweets were represented in numerical feature matrixes using word-embedding models. Then, CNN and RNN architectures were used to discriminate traffic-related tweets on the top of word vectors. Our extensive experiments clearly indicated the superiority our model performance compared to the state-of-the-art models in this domain.

However, after detecting traffic-related tweets, a unified Twitter-based traffic information system needs to identify the event location and appropriately disseminate the relevant information to end users. Assuming having access to historical tweet data and incident reports, an efficient geocoder should be developed to identify the correct location of a traffic event from its associated tweet text. Such a unified system can then be implemented in various areas of a traffic network, detect possible traffic events in a real-time basis, and finally disseminate the obtained traffic information to both drivers and traffic managers in a real-time and efficient manner, in advance to traditional news media. The disseminated information about the traffic network from Twitter data can help travelers choose the shortest and/or fastest driving routes. Also, TIM partners (e.g., transportation agencies and state police) can promptly recognize unexpected behavior in traffic flow characteristics, which helps restore smooth traffic flow as quickly and safely as possible.

Chapter 6

Conclusion

6.1 Summary of studies

In this dissertation, I proposed several deep learning architectures for addressing three transportation-related problems (1) travel mode detection, (2) vehicle classification, and (3) traffic information systems, using large-scale GPS trajectories and Twitter streaming.

With regard to the first problem, a CNN-based deep learning architecture was proposed to predict the transportation mode(s) used in an individual's trip from their raw GPS trajectories, in which modes are categorized into walk, bike, bus, driving, and train. However, travelers' raw GPS trajectories contain only a series of chronologically coordinate points without any explicit features and meaningful information. Furthermore, the structure of the raw GPS trajectory is not adaptable for deep learning algorithms. Accordingly, a novel representation for GPS trajectories was designed to not only be adaptable with the deep learning algorithms but represents fundamental motion characteristics of users. This novel representation was then passed into the proposed architecture to predict traveler's transportation mode.

Moreover, almost all the current research work on travel mode detection has built their models using only labeled trajectories. Nonetheless, a significant portion of GPS trajectories might not be annotated by a transport mode since the acquisition of labeled data is a more expensive and labor-intensive task in comparison with collecting unlabeled data. Thus, the previous research was extended by utilizing unlabeled data through the power of deep unsupervised learning algorithms such as Convolutional AutoEncoder (Conv-AE). For the first time in this domain, a deep semi-supervised architecture was developed by integrating the supervised CNN classifier and unsupervised Conv-AE to harness both unlabeled and labeled GPS

trajectories. Both components were simultaneously trained by minimizing a cost function that is a linear combination of unsupervised and supervised losses. A novel schedule for tuning the hyperparameters was also designed to connect these losses during the model training. The results of the extensive experiments revealed that the proposed GPS representation and models outperformed several supervised and semi-supervised state-of-the-art baseline methods for various amounts of labeled and unlabeled GPS trajectories.

With regard to the second problem, a deep Convolutional Neural Network for Vehicle Classification (CNN-VC) was designed so as to identify the vehicles' class from their GPS trajectories. Compared to the travel mode detection, the GPS representation was significantly improved by adding roadway features, extracted from an open source routing machine engine. Furthermore, the proposed GPS representation has enough flexibility to contain information from other external sensors (e.g., mobile phone's accelerometer and gyroscope) as well as environmental conditions. Before delving into training the CNN-VC model, an efficient programmatic strategy was also designed to label large-scale GPS trajectories (20 million) by means of vehicle information obtained from Weigh-In-Motion (WIM) systems, in which WIM is a type of fixed-point traffic flow sensors. The extensive experimental results revealed that the proposed CNN-VC model consistently outperforms both classical machine learning algorithms and other deep learning baseline methods for distinguishing between light-duty (e.g., passenger cars), medium-duty (e.g., pickups), and heavy-duty (e.g., trucks) vehicles.

With regard to the third problem, a deep-learning architecture for extracting real-time traffic congestion and incident events from Twitter streaming was developed. In order to address shortcomings in literature, word embeddings models were used to map tweets into a low-dimensional feature vector without any need of pre-defined traffic keywords. Afterward, supervised deep-learning algorithms (e.g., CNN and RNN) were trained and evaluated on the top of distributed representations of tweets, obtained from word embeddings. For training and evaluation purposes, a large volume of tweets were collected through Twitter APIs and then into three labels: (1) non-related to traffic, (2) traffic incidents, and (3) traffic condition and information. Some tweaks were also deployed to expedite the labeling process while preserving the quality of labeled data as high as possible. Experimental results on

the labeled dataset showed that the proposed deep learning approach achieves clear improvements over all state-of-the-art methods in literature for solving this problem.

6.2 Summary of contributions

In summary, this dissertation lays the groundwork on how to improve transportation models by mining spatiotemporal and crowdsourced data related to transportation domains through advanced deep learning algorithms. The major contributions of this study are summarized as follows, which interchangeably address the research questions, described in Chapter 1:

- This study proposed efficient representation(s) for spatiotemporal data with three outstanding merits: (1) adaptability with deep learning algorithms, (2) containing fundamental motion features and roadway characteristics, and (3) flexibility for involving information from other sensors.
- Depending on the type of data and applications, this study designed efficient data collection and labeling approaches at scale using big data technologies.
- This study designed and programmed a wide range of supervised and semi-supervised deep learning architectures for solving transportation-related problems. Appropriate training strategies were also proposed for further improvement in deep learning algorithms.
- The study conducted an extensive set of experiments for performance evaluation of the proposed frameworks. The experimental results clearly revealed the superiority of the proposed models compared to several supervised and semi-supervised state-of-the-art baseline methods.

6.3 Dissertation significance

For each application (i.e., travel mode detection, vehicle classification, and traffic information), this dissertation achieved to build an efficient model, based on spatiotemporal and crowdsourced information, using advances in artificial intelligence

(i.e., deep learning, natural language processing, and machine learning). Having such improved transportation models, the significance and practical usage of this dissertation in each application are summarized as follows:

6.3.1 Travel mode detection

The proposed travel mode detection brings advantages to users, transportation agencies, and application systems as follows:

- *Users.* The information on travel choice mode helps individuals effectively reflect on their past events and deeply understand their own life pattern as well. Also, it presents richer knowledge over the plain GPS tracks to other users and facilitates life sharing among people.

- *Transportation agencies.* (1) The information on travel choice mode helps to identify regions with high auto dependency and encourage public transport ridership by improving transit systems. (2) Suitable policies such as High-Occupancy-Vehicle (HOV) lanes can be taken during the peak-period congestion according to existing mode shares. (3) Since transportation mode is a type of travel behavior, it helps to mine the people's mobility behaviors and social patterns. Such information helps decision makers to provide appropriate services for citizens. (4) Identifying the distribution of transportation modes in a traffic network is one of the fundamental steps in urban planning, which can be achieved by the proposed model using GPS trajectories.

- *Application Systems.* (1) The information on travel choice mode empowers the application systems (e.g., Google maps) to distinguish GPS tracks by transportation modes, and then provide appropriate services (e.g., more accurate travel time estimation and fastest route) to users according the transportation mode that they are using. (2) The information on travel choice mode allows the systems to mine deeper knowledge such as traffic condition, popular routes for every distinct transportation mode, etc., from public GPS data.

6.3.2 Vehicle Classification

The proposed vehicle classification model can be readily used to label large-scale trajectories that transportation agencies (e.g., Department of Transportation) are using to support their analysis and decision making. Such enriching of the trajectory data would enable more comprehensive studies that require information about vehicle classes. Examples include, but are not limited to:

- Derivation of origin-destination tables for individual FHWA vehicle classes, which is especially useful for planning efforts where transportation analysts need to distinguish between travel patterns of passenger cars used mainly for commuting and commercial vehicles;
- Examining whether trajectories associated with heavy trucks are observed in downtown areas or along the routes with weight restrictions, which would indicate the need for additional enforcement in those areas in order to establish the desired level of safety;
- Analyzing whether trajectories attributed to large trucks are deviating from locations with weigh-in-motion systems, which would suggest the need for deploying mobile patrols in these regions to enforce weight limits;
- Estimation of FHWA-class-specific volumes along different road links in a transportation network, which would allow transportation analysts to more accurately measure performance as it would enable them to distinguish between vehicle-hour and truck-hour delays;
- More accurate estimation of emissions based on trajectory data, where individual tracks can now be attributed to different vehicle classes (e.g., passenger cars vs. trucks), which obviously makes a big difference in estimating transportation-related emissions.

6.3.3 Twitter-based traffic information system

Since the effectiveness of traditional incident detection systems is often limited by sparse sensor coverage and reporting incidents to emergency response systems is

labor-intensive, the proposed Twitter-based traffic information system can be considered as a fast approach with high-spatial coverage for extracting traffic incident information on both highways and arterials. The model can be implemented in various areas of a traffic network, detect possible traffic events in a real time, and finally disseminate the obtained traffic information to both drivers and traffic managers in a real-time and efficient manner, in advance to traditional news media. The disseminated information about the traffic network from the model can help travelers choose the shortest and/or fastest driving routes using their navigation systems. Also, Traffic Incident Management (TIM) partners, such as transportation agencies and state police, can promptly recognize unexpected behavior in traffic flow characteristics, which helps restore smooth traffic flow as quickly and safely as possible.

References

[1] Mariam Adedoyin-Olowe, Mohamed Medhat Gaber, Carlos M Dancausa, Frederic Stahl, and João Bártolo Gomes. A rule dynamics approach to event detection in twitter with its application to sports and politics. *Expert Systems with Applications*, 55:351–360, 2016.

[2] Farzindar Atefeh and Wael Khreich. A survey of techniques for event detection in twitter. *Computational Intelligence*, 31(1):132–164, 2015.

[3] Julian Ausserhofer and Axel Maireder. National politics on twitter: structures and topics of a networked public sphere. *Information, Communication & Society*, 16(3):291–314, 2013.

[4] Omar Bagdadi and András Várhelyi. Development of a method for detecting jerks in safety critical events. *Accident Analysis & Prevention*, 50:83–91, 2013.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Thanos Bantis and James Haworth. Who you are is how you travel: A framework for transportation mode detection using individual and environmental characteristics. *Transportation Research Part C: Emerging Technologies*, 80:286–309, 2017.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[9] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[10] Steve Capecci, Cathy Krupa, and Cambridge Systematics. Concept of operations for virtual weigh station. Technical report, United States. Federal Highway Administration, 2009.

[11] Sara Filipa Lemos de Carvalho et al. Real-time sensing of traffic information in twitter messages. 2010.

[12] Emmanouil Chaniotakis and Constantinos Antoniou. Use of geotagged social media in urban settings: Empirical evidence on its potential from twitter. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 214–219. IEEE, 2015.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[14] François Chollet et al. Keras, 2015.

[15] Benjamin Coifman and SeoungBum Kim. Speed estimation and length based vehicle classification from freeway single-loop detectors. *Transportation research part C: emerging technologies*, 17(4):349–364, 2009.

[16] Sina Dabiri and Kevin Heaslip. Developing a twitter-based traffic event detection model using deep learning architectures. *Expert Systems with Applications*, 2018.

[17] Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation research part C: emerging technologies*, 86:360–371, 2018.

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

[19] Eleonora D'Andrea, Pietro Ducange, Beatrice Lazzerini, and Francesco Marcelloni. Real-time detection of traffic from twitter stream analysis. *IEEE transactions on intelligent transportation systems*, 16(4):2269–2283, 2015.

[20] Zhen Dong, Yuwei Wu, Mingtao Pei, and Yunde Jia. Vehicle type classification using a semisupervised convolutional neural network. *IEEE transactions on intelligent transportation systems*, 16(4):2247–2256, 2015.

[21] Jiachen Du, Lin Gui, Ruifeng Xu, and Yulan He. A convolutional attention model for text classification. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 183–195. Springer, 2017.

[22] Hamid Reza Eftekhari and Mehdi Ghatee. An inference engine for smartphones to preprocess data and detect stationary and transportation modes. *Transportation Research Part C: Emerging Technologies*, 69:313–327, 2016.

[23] Naveen Eluru, Vincent Chakour, and Ahmed M El-Geneidy. Travel mode choice and transit route choice behavior in montreal: insights from mcgill university members commute patterns. *Public Transport*, 4(2):129–149, 2012.

[24] Yuki Endo, Toda Hiroyuki, Nishida Kyosuke, and Kawanobe Akihisa. Deep feature extraction from trajectories for transportation mode estimation. In *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 54–66, 2016.

[25] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[26] Shih-Hau Fang, Yu-Xaing Fei, Zhezhuang Xu, and Yu Tsao. Learning transportation modes from smartphone sensors based on deep neural network. *IEEE Sensors Journal*, 17(18):6111–6118, 2017.

[27] Tao Feng and Harry JP Timmermans. Transportation mode recognition using GPS and accelerometer data. *Transportation Research Part C: Emerging Technologies*, 37:118–130, 2013.

[28] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[29] Kaiqun Fu, Rakesh Nune, and Jason X Tao. Social media data analysis for traffic incident detection and management. Technical report, 2015.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[31] Fréderic Godin, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. Multimedia lab @ acl wnut ner shared task: Named entity recognition for twitter microposts using distributed word representations. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 146–153, 2015.

[32] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[33] I. GOODFELLOW, Y. BENGIO, A. COURVILLE, and Y BENGIO. Deep learning. 2016.

[34] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005.

[35] Yiming Gu, Zhen Sean Qian, and Feng Chen. From twitter to detector: Real-time traffic incident detection using social media data. *Transportation research part C: emerging technologies*, 67:321–342, 2016.

[36] Surendra Gupte, Osama Masoud, Robert FK Martin, and Nikolaos P Papanikolopoulos. Detection and classification of vehicles. *IEEE Transactions on intelligent transportation systems*, 3(1):37–47, 2002.

[37] Mark E Hallenbeck, Olga I Selezneva, and R Quinley. Verification, refinement, and applicability of long-term pavement performance vehicle classification rules. Technical report, 2014.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] Wenhao Huang, Guojie Song, Haikun Hong, and Kunqing Xie. Deep architecture for traffic flow prediction: deep belief networks with multitask learning. *IEEE Trans. Intelligent Transportation Systems*, 15(5):2191–2201, 2014.

[40] X. IU, Y. ZHU, and X. ZHANG. Deepsense: a novel learning mechanism for traffic prediction with taxi gps traces. *Global Communications Conference (GLOBECOM)*.

[41] Arash Jahangiri and Hesham Rakha. Developing a support vector machine SVM classifier for transportation mode identification by using mobile phone sensor data. In *Transportation Research Board 93rd Annual Meeting*, number 14-1442, 2014.

[42] Rie Johnson and Tong Zhang. Supervised and semi-supervised text categorization using lstm for region embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 526–534. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045447.

[43] M. Kafai and B. Bhanu. Dynamic Bayesian networks for vehicle classification in video. *IEEE Transactions on Industrial Informatics*, 8(1):100–109, Feb 2012. ISSN 1551-3203. doi: 10.1109/TII.2011.2173203.

[44] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[45] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

[47] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[49] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013.

[50] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Molding cnns for text: non-linear, non-consecutive convolutions. *arXiv preprint arXiv:1508.04112*, 2015.

[51] Guanyao Li, Chun-Jie Chen, Sheng-Yun Huang, Ai-Jou Chou, Xiaochuan Gou, Wen-Chih Peng, and Chih-Wei Yi. Public transportation mode detection from cellular data. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 2499–2502. ACM, 2017.

[52] Miao Lin and Wen-Jing Hsu. Mining gps data for mobility patterns: a survey. *Pervasive and mobile computing*, 12:1–16, 2014.

[53] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, Fei-Yue Wang, et al. Traffic flow prediction with big data: A deep learning approach. *IEEE Trans. Intelligent Transportation Systems*, 16(2):865–873, 2015.

[54] Nikola Marković, Przemysław Sekuła, Zachary Vander Laan, Gennady Andrienko, and Natalia Andrienko. Applications of trajectory data from the perspective of a road transportation agency: Literature review and maryland case study. *IEEE Transactions on Intelligent Transportation Systems*, (99):1–12, 2018.

[55] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[56] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[57] Heikki Mäenpää, Lobov Andrei, and L. Martinez Lastra Jose. Travel mode estimation for multi-modal journey planner. *Transportation Research Part C: Emerging Technologies*, 82:273–289, 2017.

[58] Theresa Nick, Edmund Coersmeier, Jan Geldmacher, and Juergen Goetze. Classifying means of transportation using mobile sensor data. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–6. IEEE, 2010.

[59] Emily Parkany and Chi Xie. A complete review of incident detection algorithms & their deployment: what works and what doesn't. Technical report, 2005.

[60] Joao Pereira, Arian Pasquali, Pedro Saleiro, and Rosaldo Rossetti. Transportation in social media: an automatic classifier for travel-related tweets. In *Portuguese Conference on Artificial Intelligence*, pages 355–366. Springer, 2017.

[61] Swit Phuvipadawat and Tsuyoshi Murata. Breaking news detection and tracking in twitter. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 120–123. IEEE, 2010.

[62] Nicholas G Polson and Vadim O Sokolov. Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, 79:1–17, 2017.

[63] Marc'Aurelio Ranzato and Martin Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the 25th international conference on Machine learning*, pages 792–799. ACM, 2008.

[64] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[65] Francisco Rebelo, Carlos Soares, and Rosaldo JF Rossetti. Twitterjam: Identification of mobility patterns in urban centers based on tweets. In *Smart*

Cities Conference (ISC2), 2015 IEEE First International, pages 1–6. IEEE, 2015.

[66] Mohsen Rezaie, Zachary Patterson, Jia Yuan Yu, and Ali Yazdizadeh. Semi-supervised travel mode detection from smartphone data. In *Smart Cities Conference (ISC2), 2017 International*, pages 1–8. IEEE, 2017.

[67] Sílvio S Ribeiro Jr, Clodoveu A Davis Jr, Diogo Rennó R Oliveira, Wagner Meira Jr, Tatiana S Gonçalves, and Gisele L Pappa. Traffic observatory: a system to detect and locate traffic events and conditions using twitter. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, pages 5–11. ACM, 2012.

[68] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[69] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[70] Ronald W Schafer. What is a savitzky-golay filter? *IEEE Signal processing magazine*, 28(4):111–117, 2011.

[71] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010.

[72] Przemysław Sekuła, Nikola Marković, Zachary Vander Laan, and Kaveh Farokhi Sadabadi. Estimating historical hourly traffic volumes via machine learning and vehicle probe data: A Maryland case study. *Transportation Research Part C: Emerging Technologies*, 97:147–158, 2018.

[73] Alessio Signorini, Alberto Maria Segre, and Philip M Polgreen. The use of twitter to track levels of disease activity and public concern in the us during the influenza a h1n1 pandemic. *PloS one*, 6(5):e19467, 2011.

[74] Matteo Simoncini, Leonardo Taccari, Francesco Sambo, Luca Bravi, Samuele Salti, and Alessandro Lori. Vehicle classification from low-frequency GPS data with recurrent neural networks. *Transportation Research Part C: Emerging Technologies*, 91:176–191, 2018.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pages 151–161. Association for Computational Linguistics, 2011.

[77] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, Mike Y Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, William G Griswold, and Eyal De Lara. Mobility detection using everyday gsm traces. In *International Conference on Ubiquitous Computing*, pages 212–224. Springer, 2006.

[78] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[79] Leon Stenneth, Ouri Wolfson, Philip S Yu, and Bo 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*, pages 54–63. ACM, 2011.

[80] Zhanbo Sun and Xuegang Jeff Ban. Vehicle classification using GPS data. *Transportation Research Part C: Emerging Technologies*, 37:102–117, 2013.

[81] Joseph S Sussman. *Perspectives on intelligent transportation systems (ITS)*. Springer Science & Business Media, 2008.

[82] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[83] Charles Truong, Laurent Oudre, and Nicolas Vayatis. A review of change point detection methods. *arXiv preprint arXiv:1801.00718*, 2018.

[84] Thaddeus Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey review*, 23(176):88–93, 1975.

[85] Hao Wang, Liu GaoJun, Duan Jianyong, and Zhang Lei. Detecting transportation modes using deep neural network. In *IEICE Transactions on Information and Systems*, pages 1132–1135, 2017.

[86] Napong Wanichayapong, Wasawat Pruthipunyaskul, Wasan Pattara-Atikom, and Pimwadee Chaovalit. Social-based traffic information extraction and classification. In *ITS Telecommunications (ITST), 2011 11th International Conference on*, pages 107–112. IEEE, 2011.

[87] Billy M Williams and Angshuman Guin. Traffic management center use of incident detection algorithms: Findings of a nationwide survey. *IEEE Transactions on intelligent transportation systems*, 8(2):351–358, 2007.

[88] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. Modeling trajectories with recurrent neural networks. IJCAI, 2017.

[89] Linlin Wu, Biao Yang, and Peng Jing. Travel mode detection based on GPS raw data collected by smartphones: a systematic review of the existing methodologies. *Information*, 7(4):67, 2016.

[90] Yuankai Wu and Huachun Tan. Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. *arXiv preprint arXiv:1612.01022*, 2016.

[91] Zhibin Xiao, Yang Wang, Kun Fu, and Fan Wu. Identifying different transportation modes from trajectory data using tree-based ensemble classifiers. *ISPRS International Journal of Geo-Information*, 6(2):57, 2017.

[92] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[93] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[94] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. Trajectory clustering via deep representation learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 3880–3887. IEEE, 2017.

[95] Yizhe Zhang, Shen Dinghan, Wang Guoyin, Gan Zhe, Henao Ricardo, and Cari Lawrence. Deconvolutional paragraph representation learning. In *Advances in Neural Information Processing Systems*, pages 4172–4182, 2017.

[96] Yu Zheng. Trajectory data mining: An overview. *ACM Trans. Intell. Syst. Technol.*, 6(3):29:1–29:41, May 2015. ISSN 2157-6904. doi: 10.1145/2743025. URL http://doi.acm.org/10.1145/2743025.

[97] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 312–321. ACM, 2008.

[98] Yu Zheng, Like Liu, Longhao Wang, and Xing 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*, pages 247–256. ACM, 2008.

[99] Yu Zheng, Fu Hao, Xie Xing, Ma Wei-Ying, and Li Quannan. Geolife GPS trajectory dataset-user guide. 2011.

[100] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):38, 2014.

[101] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.