A Deep Learning Model for Transportation Mode Detection Based on Smartphone Sensing Data

Xiaoyuan Liang, Yuchuan Zhang, Guiling Wang, and Songhua Xu

***Abstract*— Understanding people's transportation modes is beneficial for empowering many intelligent transportation systems, such as supporting urban transportation planning. Yet, current methodologies in collecting travelers' transportation modes are costly and inaccurate. Fortunately, the increasing sensing and computing capabilities of smartphones and their high penetration rate offer a promising approach to automatic transportation mode detection via mobile computation. This paper introduces a light-weighted and energy-efficient transportation mode detection system using only accelerometer sensors in smartphones. The system collects accelerometer data in an efficient way and leverages a deep learning model to determine transportation modes. Different architectures and classification methods are tested with the proposed deep learning model to optimize the system design. Performance evaluation shows that the proposed new approach achieves a better accuracy than existing work in detecting people's transportation modes.**

***Index Terms*— Transportation mode, deep learning, smartphone, accelerometer.**

I. Introduction

THE increasing sensing and computing capabilities of smartphones offer a promising new approach to monitoring human activities [1], [2], including means to detect travelers' transportation modes, which are particularly important for developing many transportational applications. For example, (1) knowledge of people's transportation modes is useful for improving urban transportation planning [3], [4]. The new method would transform the way how transportation demand information is gathered for supplementing the traditional information acquisition practice based on telephone interviews and questionnaires, which is expensive and time consuming to conduct [5]. (2) The knowledge can also improve the performance of localizing and positioning systems. With the awareness of transportation modes, localizing systems can more precisely narrow down users' locations [3]. (3) The information facilitates targeted and customized advertisements and services [5] based on the transportation modes the users are taking. (4) The acquired information can also help improve smartphone users' physical habits for environment protection purpose. For example, the CO_2 footprint as well as the calories burned by individuals can be better monitored with the information [4], [6]. In this way, the data can help users build green transportation habits to protect the environment.

Most previous studies use data from Global Positioning System (GPS) to detect people's transportation modes [5]–[7]. However, GPS-based methods suffer from the following drawbacks [3], [4]: (1) GPS signals are not available everywhere. GPS requires an unobstructed view to satellites, limiting its applicability in metropolitan areas with highrises or in shielded areas; (2) a GPS sensor consumes a significant amount of energy and may rapidly deplete the battery of a mobile device. To address these issues, some existing work uses alternative sensors to detect transportation modes. For example, Jahangiri and Rakha [4] propose leveraging an accelerometer coupled with a gyroscope and a rotation vector sensor to detect five transportation modes. Fang *et al.* [8] use a deep neural network to classify five transportation modes based on the data from the accelerometer, magnetometer and gyroscope. Hemminki *et al.* [3] propose using an accelerometer sensor to detect six transportation modes. However, the detection accuracy of the above works, less than 90%, still needs improvement when only using the accelerometer sensor. In contrast, our proposed approach can detect all common transportation modes, including being stationary, walking, bicycling, taking bus, driving a car, taking subway, and taking train, using data only acquired from accelerometer sensors in smartphones. The key design objectives of our new work include low energy consumption, applicability in all situations and detection accuracy.

To achieve aforementioned three objectives, we propose a deep learning framework to efficiently detect a user' transportation mode using only data read from her smartphone's accelerometer sensor. Previous studies [8]–[10] using deep learning in transportation mode detection usually involve other sensors to extract trajectory data, which are energy-consuming, and mainly employ dense networks and recurrent neural networks. In our model, we only take the accelerometer data and design a convolutional neural network on one-dimension data to accurately detect the transportation modes. The data are processed by removing gravity and smoothing. To minimize the influence from different axes and rotation, we also use the magnitude instead of the value in every axis to develop the model, such that the detection accuracy will not be affected

Manuscript received February 3, 2019; revised July 28, 2019; accepted October 2, 2019. This work was supported in part by the National Science Foundation under Grant CMMI-1844238. The Associate Editor for this article was F. Chu. *(Corresponding author: Xiaoyuan Liang.)*

X. Liang and G. Wang are with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: xl367@njit.edu; gwang@njit.edu).

Y. Zhang is with the Big Data Engine Department, MicroStrategy, Annandale, VA 22182 USA (e-mail: yz596@njit.edu).

S. Xu is with the Department of Information System, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: songhua.xu@njit.edu).

Digital Object Identifier 10.1109/TITS.2019.2951165

by the position of a smartphone relative to its user considering a user may hold her phone in any orientation. The data are divided into small windows. Data in every window are fed into our proposed deep learning model to detect the window's corresponding transportation mode. Traditional classification models in machine learning are trained with features from both time and frequency domains as baselines. Simulation results show that the proposed system can achieve a higher detection accuracy than all peer methods.

To summarize, our work makes the following contributions. (1) To the best of our knowledge, this is the first attempt to adopt a deep convolutional neural network framework on the accelerometer data only to detect transportation modes in near realtime, with about 1 second delay. (2) Different from many existing systems, our system can be widely applied since we only use data read from accelerometer sensors which are available in any smartphones. The system is energy efficient by only using the accelerometers for they consume much less energy than other motion sensors [11]. Comparing to the works that only use accelerometers, our system is robust to the position and orientation of smartphones which greatly improves user experience, since in our work the magnitude is used instead of the vectors in three axes. (3) It is shown that the proposed model outperforms existing studies and can accurately detect transportation modes via extensive experiments in which the same window size is taken on the same dataset. (4) The data are collected from the accelerometer in smartphones under seven different transportation modes. An anonymized dataset will be made public for peer researchers[1].

The remainder of the paper is organized as follows. Section II introduces our system architecture. Section III details data collection and data preprocessing in our system. Our deep learning model is introduced in Section IV. Section V evaluates our system in terms of prediction accuracy and compares our model with its variations. Section VI presents related work and compares our system with them. The paper concludes in Section VII with discussion on future research directions.

II. Architecture

The objective of our paper is to design a system to efficiently detect a user's transportation mode in real time using only data read from her smartphone's accelerometer sensor. The architecture of our system is illustrated in Fig. 1. As shown in the figure, our system has three components: data collection, data preprocessing, and deep learning model development using the processed data. (1) Data collection. We compare different sensors of a smartphone and determine the accelerometer data are to be collected. We develop an Android application to collect the data. (2) Data preprocessing. The raw data are cleaned and processed to remove the impact of the gravity, to be smoothed, and to be transferred into one-dimension segments, before being used to construct a detection model. (3) Deep learning model development and training. We develop a Convolutional Neural Network (CNN) on one-dimension data for transportation mode detection. After being trained by the processed data, the model reports the corresponding transportation mode in real time, given any new incoming accelerometer data.

[1]The data used in this paper can be downloaded from https://cs.njit.edu/∼gwang/TITS2019.html.

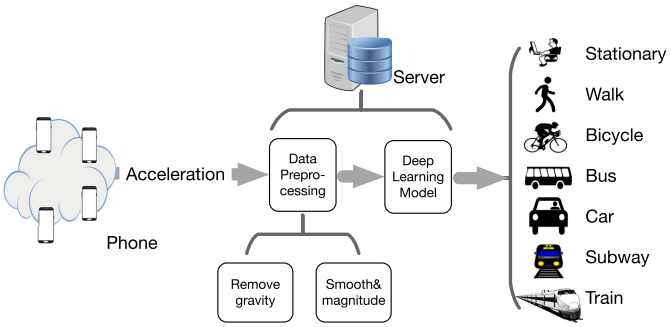


Fig. 1. System architecture.

III. Data Collection and Preprocessing

A. Data Collection

We choose to accelerometer as data source to detect transportation modes for its energy-efficiency. The sensor consumes 10 times less power than other motion sensors [11], e.g., gyroscope. To collect accelerometer data, an Android application is developed. A phone is carried under different transportation modes and the collected data are manually labeled with the corresponding transportation mode by the corresponding travelers. When collecting the data, the phone can be placed as usual and is not required to stay in a specific position or orientation. Note that this is different from many previous work, which requires that the phone must be placed in the hip pocket [6], or it has to be kept in a bag [3]. Loosing the requirement improves user experience, but poses new challenges in processing the data to remove noises.

In previous studies, the acceleration sampling frequency ranges from 25Hz [6] to 100Hz [4], [12]. In this paper, we choose a middle sampling frequency, 50Hz, which is the same as the one adopted in the prior study [13]. The frequency can balance the information precision and the energy consumption.

The original data acquired from an accelerometer are organized as three dimensional vectors, where each vector component corresponds to the value in one axis in the mobile phone's coordinate system. The coordinate system is shown in Fig. 2 [14]. One piece of sample data is a vector with one value per axis in the coordinate system in the unit of m/s^2.

B. Preprocessing

The collected acceleration data contain a gravity component, which is generated by the earth and everyone on the earth is influenced, so this component does not contribute to transportation mode detection. Thus, the first step of preprocessing is to remove the gravity component so that the remaining part only carries characteristics of different transportation modes. The data are then smoothed to remove large fluctuations, which may be caused by sudden movements. For example,

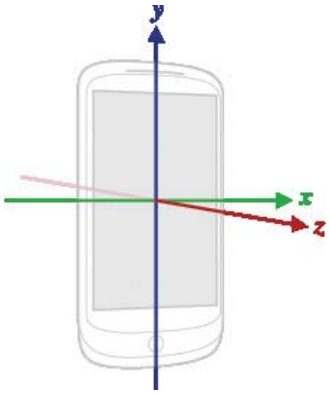


Fig. 2. The coordinate system of a smartphone. The x-axis is from left to right, the y-axis is from down to up, and the z-axis is from back to front.

a walking person may suddenly stop to check his beeping phone and then continue walking. Finally, the acceleration magnitude is obtained to build the classification model.

1) Removing Gravity: Gravity is reflected in all three axes in the smartphone?s coordinate system, which is not possible to be removed by subtracting a constant value. When a phone is put on a stationary desk with the screen up, the acceleration value is expected to be $(0, 0, -g)$, where g is the gravity constant. However, a phone can be placed in any direction, so even when the phone is stationary, the first two dimensions of the data may not be zero and the last one may not be $-g$. Generally, all the three axes contains a portion of the gravity. Removing the gravity results in a new movement record, which is used for actual transportation mode detection. The acceleration data obtained after removing the gravity part are called linear acceleration data.

As for how to remove the gravity component, a low-pass filter is applied to remove gravity [15], since the gravity is much more stable than the acceleration generated by movement during a relative long period of time [11], [16]. Let $\mathbf{A}_k$ denote the vector collected from the accelerometer and $\mathbf{G}_k$ denote the gravity vector at the k^{th} time point in the mobile phone's coordinate system. Let $\mathbf{L}_k$ denote the linear acceleration data after removing the gravity component. The gravity is estimated by the following equation [11], [16],

$$\mathbf{G}_k = \alpha \cdot \mathbf{G}_{k-1} + (1-\alpha)\mathbf{A}_k, \tag{1}$$

where α is the exponential decay weight between old estimated gravity and the new one. An empirical value, 0.8, is taken in the system [14]. $\mathbf{L}_k$ is calculated as follows,

$$\mathbf{L}_k = \mathbf{A}_k - \mathbf{G}_k. \tag{2}$$

The linear acceleration data is the collected acceleration data minus the estimated gravity in three axes. The value of G_k is initialized to be the readings from the accelerometer.

2) Data Smoothing: Data smoothing is necessary since a phone's movement is not always consistent with its user's movement and the inconsistent part needs to be removed as much as possible. There are two causes of the inconsistency. One is the sudden movement of a phone irrelevant to its user's movement. For example, a user may suddenly pick up the phone while driving or in other transportation modes.

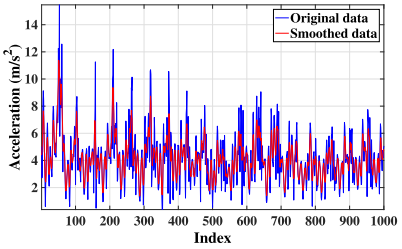


Fig. 3. The smoothed signals in the time domain.

In a sudden movement like this, the acceleration changes abruptly. The data do not reflect its user's movement, and thus impair the accuracy of transportation mode detection. Therefore, the data are smoothed to reduce the influence by those sudden movements.

The data are smoothed by the central moving average algorithm, which is a special Savitzky-Golay filter [17]. It is calculated by averaging an odd number of nearest neighbors, m, in the time series. m is a predefined constant value, which is set to 5 in this paper. For the original acceleration at the kth time point, $\mathbf{L}_k$, let $\hat{\mathbf{L}}_k$ denote the corresponding estimated linear acceleration value after smoothing. K is the total number of vectors in the data. $\hat{\mathbf{L}}_k$ is estimated as follows,

$$\hat{\mathbf{L}}_k = \begin{cases} \dfrac{\sum_{i=1}^{2k-1} \mathbf{L}_i}{2k-1} & k \in [1, \lfloor \frac{m}{2} \rfloor], \\ \dfrac{\sum_{i=k-\lfloor m/2 \rfloor}^{k+\lfloor m/2 \rfloor} \mathbf{L}_i}{m} & k \in (\lfloor \frac{m}{2} \rfloor, K - \lfloor \frac{m}{2} \rfloor), \\ \dfrac{\sum_{i=2k-K}^{K} \mathbf{L}_i}{2(K-k)+1} & k \in [K - \lfloor \frac{m}{2} \rfloor, K]. \end{cases} \tag{3}$$

If the number of neighbors in one side is less than K, then the average algorithm takes the same maximum possible number of neighbors from both sides.

Take Fig. 3 as an example to show the effect after smoothing the data. The data is from a stable period of walking. In the figure, the blue line shows the original data and the red line shows the smoothed data after averaging the nearest 5 neighbors. The x-axis is the index of data and the y-axis is the acceleration amplitude. From this figure, we can see that the data becomes less fluctuating than the original one. It can effectively reduce the large fluctuation and sudden movements.

3) Taking Magnitude and Data Slicing: Inconsistency aforementioned also comes from the fact that a phone's orientation is likely to be different from that of its user. The phone may be put in any position and orientation and they are subject to changes at times. Thus, we take the magnitude of the acceleration data. Even though there may be some information loss, the magnitude is more robust to a phone's changing and unpredictable orientation according to prior studies [18]. For $\hat{\mathbf{L}}_k = (\hat{L}_{k_x}, \hat{L}_{k_y}, \hat{L}_{k_z})$, we use $|\hat{\mathbf{L}}_k|$ to denote its magnitude, which is calculated as,

$$|\hat{\mathbf{L}}_k| = \sqrt{\sum_{i=x,y,z} \hat{L}_{k_i}^2}. \tag{4}$$

To conduct real-time detection, the time series data are divided into small windows. The system detects every separate

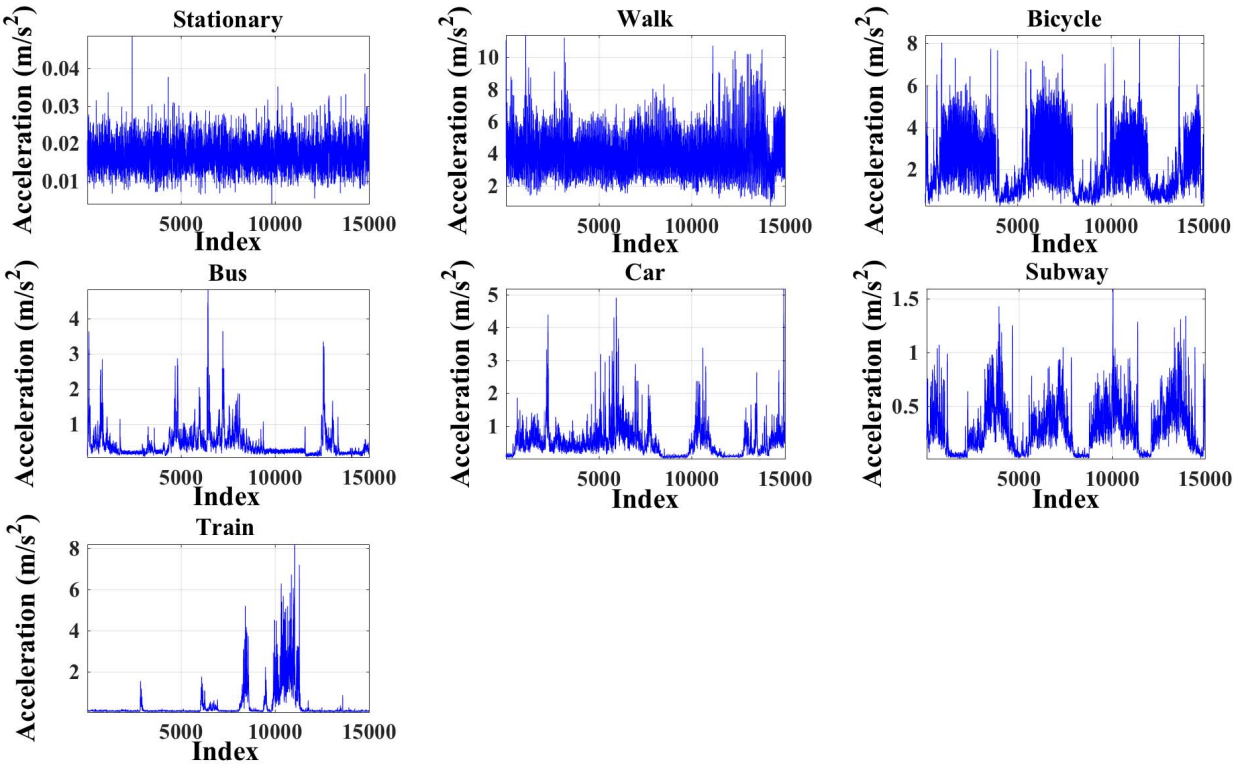


Fig. 4. The acceleration magnitude of seven transportation modes.

window's transportation mode. Specifically, the data are grouped into a window of samples with a window size of S seconds and a sliding size of s seconds. By default, S is set to 10.24 seconds (512 samples) and s is set to 1.28 seconds (64 samples). We choose the two values because the frequency transformation in traditional machine learning methods requires the number of samples to be the exponentiation of 2 [19]. With the selected values, the transportation mode is detected almost every second based on a 10-second historical information. We test different values of S and s in the experiments to examine their impact on performance.

C. Preliminary Data Analysis

We plot the acceleration magnitude of different transportation modes during a selected period in Fig. 4. The x-axis is data index and the y-axis is the acceleration magnitude. From the figure, we can see that the acceleration data in different modes show different patterns. For example, the acceleration data in bicycling shows an obvious periodicity and the data in walk have the largest acceleration values compared with others. The acceleration value in the stationary mode is the smallest. The figure also shows that the data in the bus and car modes are similar in the shape as both modes keep low acceleration at most time and have a few rapid changes.

We also plot the corresponding data in the frequency domain as a time-frequency plot in Fig. 5. The data are sampled every 128 values with 64 values overlapping with neighbors. It means that the window is 2.56 seconds and the overlap is 1.28 seconds. In the figure, the x-axis is the frequency and y-axis is the time. The color shows the magnitude, which is the same as the log-scale power spectral density. From the figure, we can see that most power is gathered in the frequency domain less than 10Hz. It is reasonable for us to select the power density from the frequency at 1Hz to 10Hz as features for the traditional machine learning methods.

Note that we choose seven traditional machine learning models as classification methods to determine transportation modes as a benchmark to our deep learning model. The background of traditional machine learning methods employed for transportation mode detection is presented in Appendix B.

IV. Deep Learning Model

The deep learning model adopted in our system is a deep Convolutional Neural Network (CNN). CNNs, as one special type of deep learning models, are commonly used to recognize objects in image processing. In this paper, a CNN is built on the one-dimension acceleration data to determine the transportation mode in every time window. We present the background of convolution, max-pooling and full-connection of the CNN used in our approach in Appendix A. In this section, we first present our deep learning model's network architecture. Then we introduce the elements of our model, including the nonlinear function employed in every layer, the loss function in evaluating the performance of a classification model, the optimization method used to minimize the loss function, and the normalization on the data to improve performance.

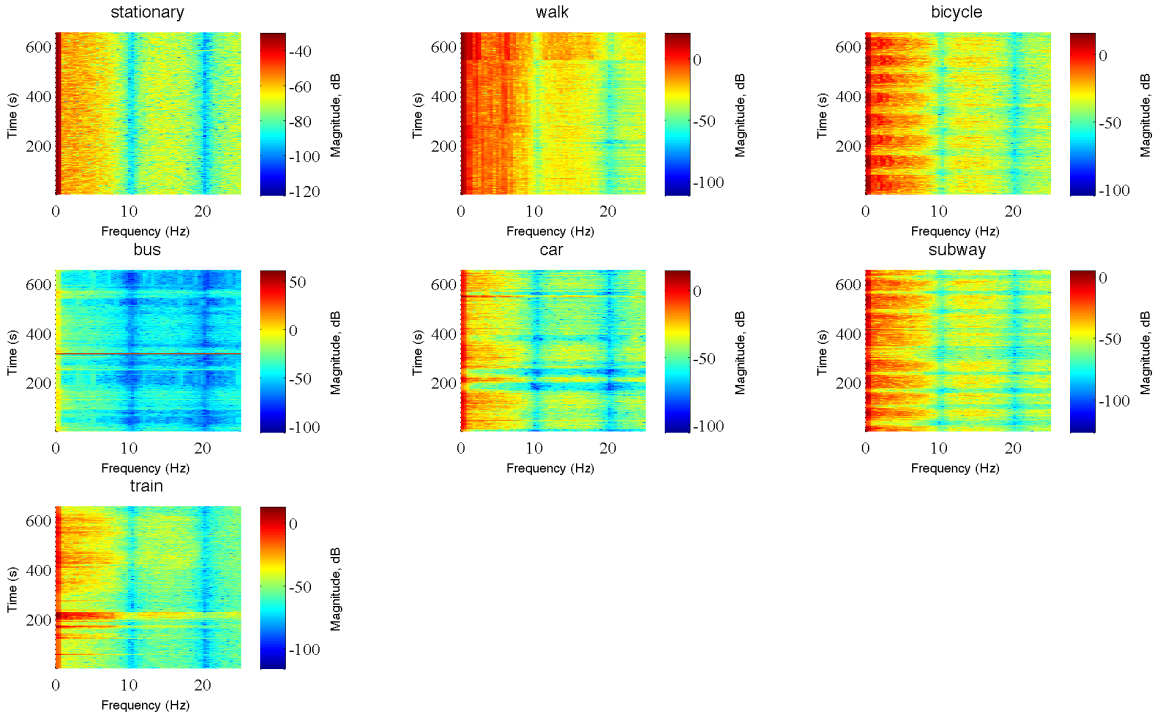


Fig. 5. Time-frequency domain figures of different transportation modes. The x-axis is the frequency, the y-axis is the time and the color denotes the magnitude.

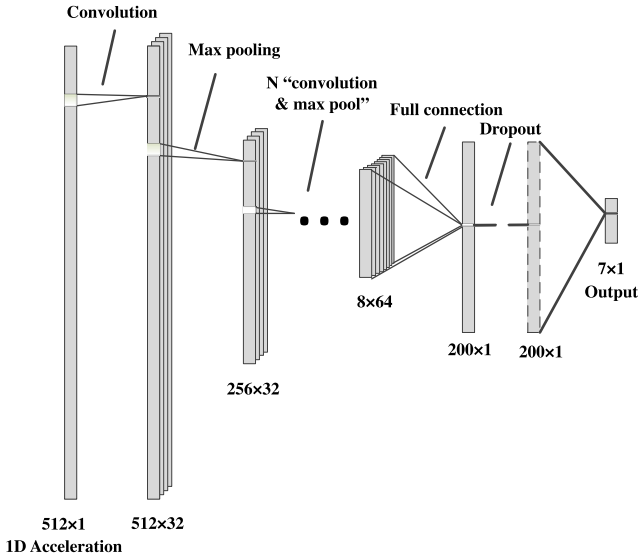


Fig. 6. The CNN architecture in our system.

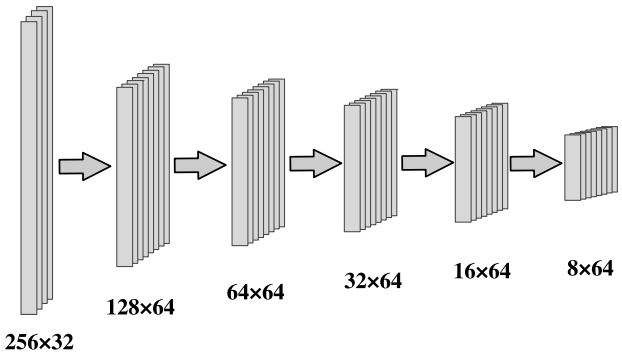


Fig. 7. The hidden N layers in Fig. 6.

1) Network Architecture: Our CNN is designed for one-dimensional data with the architecture shown in Fig. 6, which is inspired by AlexNet [20]. The network takes the one-dimensional acceleration data in a window as input and outputs the probability of each transportation mode in the window. The network consists of a succession of convolutional, max pooling and fully-connected layers, whose specification is as follows.

The input feature is a 512 $\times$ 1 vector of the magnitude of the acceleration data. The first dimension is the temporal space in the window and the second dimension is the size of values at every temporal point. The first convolutional layer has 32 filter banks with the 15$\times$1 shape and the filter bank moves with a stride of 1 feature at a time. The following max pooling layer is set to a 4-feature window and a stride of 2 features. The convolutional layer and max pooling layer are repeated N times. In the proposed network, N is set 6. All the max pooling layers are configured using the same parameters. The second and third convolutional layers have 64 filter banks on a 10-feature window with a stride size of 1 feature. The other four convolutional layers filter the data with 64 filter banks on a 5-feature window with a stride of 1 feature. After the 6 times of convolution and max pooling processes, the data become 8 $\times$ 64 dimension, as shown in Fig. 7. The data are then fed into a fully-connected layer and become 200 $\times$ 1. A dropout layer is not employed in our system, but it is added after the fully-connected layer only for comparison, which is used to evaluate whether the dropout layer can help improve the performance. Finally, the data are transferred into

a 7×1 output vector with full connection. The value in the output is the probability for every transportation mode.

2) Nonlinear Function: In a neural network, the common ways to model a neuron's output f as a function of its input x are tanh, sigmoid or Rectified Linear Unit (ReLU). ReLU outperforms the other two with its simple form ($f(x) = max(0, x)$) in the fast convergence of stochastic gradient descent [20], but it may generate 'dead' neurons, which are never activated. In the proposed model, a leaky ReLU [21] is employed, which is defined as follows:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \beta x, & \text{if } x \leq 0 \end{cases}. \tag{5}$$

β is a small constant to avoid zero gradient in the negative side. It should a very small value to make the negative values propagate and the positive values dominate the propagation. In the experiments, it is shown that the leaky ReLU can converge faster than the traditional ReLU. The nonlinear activation function is attached to every convolutional layer and the fully-connected layer.

3) Loss Function: A loss function is used to evaluate how labels derived by the current classification model deviate from the corresponding true labels. The loss function is usually composed of two parts, the variance between the estimated labels and true labels, and the regularization. The loss L is shown in the following equation,

$$L = V + \lambda \cdot R, \tag{6}$$

where V denotes the variance and R is the regularization value. λ is the weight decay determining how much the regularization affects the final loss.

The variance is expressed by the softmax cross entropy, as follows:

$$V = -\sum_{id}\sum_{c} t_{id,c} \cdot log(y_{id,c}), \tag{7}$$

where id denotes the index of a sample in the mini-batch and $y_{id,c}$ is the value of sample id at class c in the output layer. If the sample belongs to class c, the value of $t_{id,c}$ is 1; otherwise, it is 0. If an estimated label is the same as the true one, the contribution to the final V from the sample is 0; if the estimated label and the true one are different, the contribution becomes very large.

The regularization is to avoid overfitting during model training [22]. The proposed neural network contains over 100,000 parameters. To reduce overfitting, there are two possible ways, regularization [22] and dropout [20]. Regularization is to decrease the scale of a neural network by making weights as close to zero as possible. Dropout is to randomly make some weights as zero to increase the network's robustness. In this network, the L2 regularization is employed in the loss function. In L2 regularization, the value of R is the squared sum of all weights in the CNN.

4) Optimization: Gradient descent is one of the most popular algorithms to optimize the loss function in neural networks. Among all variants of gradient descent algorithms, Adaptive Moment Estimation (Adam) [23] is favorably reviewed due to its capability for attaining satisfactory overall performance with a fast convergence and adaptive learning rate [24]. The Adam optimization method adaptively updates the learning rate considering both first-order and second-order moments using the stochastic gradient descent procedure. Specifically, let $\boldsymbol{\theta}$ denote the parameters in the CNN and $L(\boldsymbol{\theta})$ denote the loss function. Adam first calculates the gradients of the parameters,

$$\mathbf{g} = \nabla_{\theta} L(\boldsymbol{\theta}). \tag{8}$$

It then respectively updates the first-order and second-order biased moments, $\mathbf{s}$ and $\mathbf{r}$, by the exponential moving average,

$$\begin{aligned} \mathbf{s} &= \rho_s \mathbf{s} + (1 - \rho_s)\mathbf{g}, \\ \mathbf{r} &= \rho_r \mathbf{r} + (1 - \rho_r)\mathbf{g}, \end{aligned} \tag{9}$$

where ρ_s and ρ_r are the exponential decay rates for the first-order and second-order moments, respectively. The first-order and second-order biased moments are corrected using the time step t through the following equations,

$$\begin{aligned} \hat{\mathbf{s}} &= \frac{\mathbf{s}}{1 - \rho_s^t}, \\ \hat{\mathbf{r}} &= \frac{\mathbf{r}}{1 - \rho_r^t}. \end{aligned} \tag{10}$$

Finally the parameters are updated as follows,

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \\ &= \boldsymbol{\theta} + (-\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}), \end{aligned} \tag{11}$$

where ϵ is the initial learning rate and δ is a small positive constant to attain numerical stability.

5) Normalization: The classification results can be improved through normalization. Assume the dataset is not too large to store in the memory. The whole training dataset is used to normalize the whole dataset. The mean μ and variance σ of the training dataset are derived first. The normalized data x' from the original x is calculated through [25],

$$x' = \frac{x - \mu}{\sigma}. \tag{12}$$

The normalization process is done before the data is used to train the CNN.

V. System Implementation and Performance Evaluation

Key experimental results are presented in this section with performance attained by the method. The implementation details are first presented, and then the data are analyzed. Finally, the detection performance of the system and the comparison with other methods are presented in detail.

A. System Implementation

We developed an Android application and installed it on a Google Nexus 5X smartphone and a Google Nexus 6 smartphone, which are shared by four different users at different time. This is due to the size limit of our research group. The relative small number of participates helps in maintaining the high quality of the collected data because some data need to be manually input. Users can freely hold the phone in any

TABLE I
PARAMETERS USED IN THE CONVOLUTIONAL NEURAL NETWORK

Parameter	Value
Minibatch size	100
L2 regularization λ	0.001
Leaky ReLU β	0.01
1st-order moment weight ρ_s	0.9
2nd-order moment weight ρ_r	0.999
Learning rate ϵ	0.0001
Constant δ	10^{-8}

orientation to their preference. The orientation and placement of mobile phones are not restricted in the system. This scenario can help keep our data in high quality and large variety. The Android application records the accelerometer data at 50Hz and the current transportation mode is manually labeled by the users. The acceleration data are collected in the following seven transportation modes: walking, bicycling, taking bus, driving a car, taking subway, taking train and being stationary. The data in every mode are collected for about 2 hours. Specifically, users take train from New Jersey to Washington, D.C. to collect the transportation mode of taking a train; they take the subways in New York city; they drive cars in local roads and on highways around NJIT campus; they take bus between home and NJIT campus; they walk around campus and in nearby parks; the stationary mode is sampled on NJIT campus.

Matlab is used to preprocess the data and the CNN is built using Tensorflow [26] on one NVIDIA GTX 1050 Ti 4GB GPU. The traditional models are built and tested in Weka [27].

In terms of the setting of hyperparameters, by default, a 512-sample window moves 64 samples every time to generate a new window data (512 and 64 are chosen for generating frequency values in traditional machine learning models, which are not required in the CNN). It means one output is generated every 1.28 seconds based on about 10-second historical values. Other parameters used in our CNN are shown in Table I. The minibatch size is chosen to balance the computation cost and convergence speed. A larger minibatch can quicken the convergence, but involves more computation especially for GPUs. Due to the limit of computation resource, we set a relative small number, 100, to balance the two parts. The value of β is set 0.01, which is a suggested value in the paper that proposed it [21]. The value of λ is chosen from $\{10^{-2}, 10^{-3}, 10^{-4}\}$ and 10^{-3} is chosen because it can achieve the best performance. The other parameters, 1st-order moment weight ρ_s, 2nd-order moment weight ρ_r, learning rate ϵ, and constant δ, are chosen based on the suggested values from the paper that proposed the Adam optimization algorithm [23]. In the experiments, the data are primarily split into 80% as training and 20% as testing sets.

B. Evaluating Our CNN

In our test set, every transportation mode has 600 samples. We compare the classification results from our CNN with the ground truth and show the result matrix in Table II.

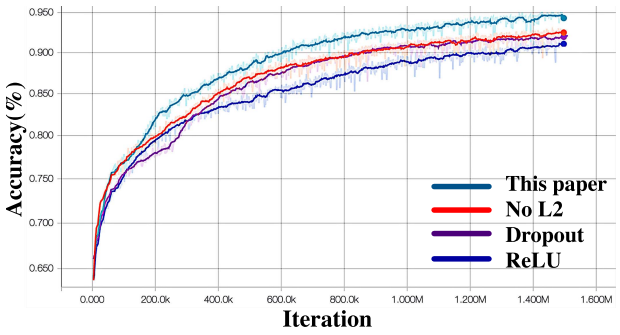


Fig. 8. The testing accuracy changes during training period. The x-axis is the training iteration index and the y-axis is the accuracy.

The rows represent the ground truth transportation modes and the columns are the transportation modes predicted by our CNN. We can see from the table that our CNN can achieve the accuracy of 94.48% on average. The result also shows that it is hard to classify the motorised transportation modes, like car, bus and subway. In those related works, if they do not try to detect bus or cars, the accuracy can be higher.

We also evaluate different CNN setups and illustrate the resulted in Table III. The first row shows the results from the employed architecture of CNN in our system, which uses Leaky ReLU as the activation function, uses L2 regularization, and does not employ dropout. The other rows are the variations of our CNN. In the compared CNNs, one part of the proposed CNN is modified to get a new CNN model. All CNNs are trained 1.5 million batch iterations with 100 windows in one batch. The results show that the proposed CNN outperforms all the other three. Specifically, our CNN achieves an accuracy of 94.48%.

The accuracy changes in the four CNNs during the training are shown in Fig. 8. In the figure, the x-axis shows the iteration index and the y-axis indicates the testing accuracy. The data in the figure are smoothed with a parameter 0.5. Shown in the figure, the proposed CNN outperforms the others and converges faster than the others.

C. Comparison With Traditional Machine Learning Methods

We compare the performance of our proposed CNN model and other traditional machine learning methods under different window sizes in this section. (The background of traditional methods is presented in Appendix B.) One special method is a Recurrent Neural Network (RNN), a Long-Short Term Memory (LSTM) [28]. We use two LSTM layers with 32 units in each layer and the other hyper-parameters follow the standard way. A dense layer with softmax activation function is stacked on the top. The optimization algorithm and batch size are the same as those in our CNN model. The window size changes from 128 to 512 with 64 distinguishing values between two adjacent windows. It means the time length in a window is from 2.56 to 10.24 seconds. The classification results under different classification models with different window sizes are shown in Table IV. The table shows that our CNN outperforms all the other methods under all window sizes. Among all

TABLE II
CLASSIFICATION MATRIX

		Classified							
		stationary	walk	bicycle	bus	car	subway	train	accuracy (%)
Ground Truth	stationary	598	2	0	0	0	0	0	99.7
	walk	3	558	0	7	4	12	14	93.0
	bicycle	0	0	592	2	6	0	0	98.7
	bus	0	6	0	554	28	12	0	92.3
	car	0	3	6	30	533	26	2	88.9
	subway	1	19	0	12	33	535	0	89.2
	train	0	0	0	0	3	0	597	99.5
average accuracy									94.48

TABLE III
CNNS' CLASSIFICATION RESULTS

Activation function	Regularization	Dropout	Accuracy (%)
Leaky ReLU	L2	No	**94.48**
Leaky ReLU	No	No	92.63
Leaky ReLU	L2	0.9	91.90
ReLU	L2	No	91.87

TABLE IV
CLASSIFICATION RESULTS COMPARING WITH TRADITIONAL MODELS UNDER DIFFERENT WINDOW SIZES

Algorithm	Window size		
	128	256	512
Naive Bayes	58.34%	59.87%	60.79%
Bayes Network	62.47%	65.30%	68.77%
Decision Tree	67.34%	72.96%	81.96%
K Nearest Neighbor	66.48%	69.07%	76.03%
Random Forest	74.09%	79.68%	90.11%
Adaptive Boosting	65.63%	71.65%	80.73%
Neural Network	69.14%	70.43%	76.30%
Supporting Vector	73.26%	75.76%	84.80%
LSTM	66.36%	60.69%	58.81%
CNN	75.48%	82.42%	**94.48%**

traditional methods, random forest performs best in accuracy by assembling multiple decision trees and different features. The proposed CNN outperforms random forest by 94.48% to 90.11% when the window size is 512. In addition, for any particular classification algorithm, the larger the window size is, the higher the accuracy can be reached. It means when the window size is too small, much information cannot be covered in the window. For example, a sudden stop while driving may be classified as stationary when the window size just catches the stop period. The reason why a LSTM does not work well in this scenario is that the number of time steps is too large (if the window size is 512, the number of time steps is 512) and there is only one value at every time step. Thus, it is hard to learn a converged model by directly applying a LSTM.

VI. COMPARING WITH RELATED WORK

In this section, we present related work and compare the performance of some of them with our model. There are several studies using accelerometers only to detect transportation modes [3], [6], [13], [29]. Hemminki *et al.* [3] collect accelerometer data at the frequency from 60 to 100Hz and divide the data into 1.2-second windows with 50% overlap. They extract 27 features in every window and train an adaptive boosting to classify the data into six modes, stationary, walk, bus, train, metro and tram. They achieve an accuracy of 80.1%. Manzoni *et al.* [6] collect accelerometer data at the frequency of 25Hz and divide it into windows of 10.24 seconds length with 50% time overlap. They also extract features from the FFT coefficients in every window and train a decision tree to classify the data into eight modes: walk, bicycle, bus, car, metro, train, still, and motorcycle. They achieve an accuracy of 82.14%. Yang [29] collects accelerometer data at the frequency of 36Hz and divide the data into 10-second windows with 50% overlap. Features are extracted from time and frequency domains and a decision tree is used to classify six transportation modes, sitting, standing, walk, run, bicycle, and car. The accuracy is 90.6%. Table V shows a summary of the three studies using only the acceleration data. Compared with the existing studies using acceleration data, the proposed system can provide higher accuracy. Existing studies usually use the traditional machine learning methods to detect transportation methods. In our work, we have shown that CNNs outperform traditional machine learning methods in detecting transportation modes. In addition, among the traditional methods, random forest performs best in the accuracy metric instead of other ones used in the existing works.

In order to make these methods [3], [29] comparable with our model, we reproduce these studies by extracting the same features and applying the same machine learning methods shown in the original papers. The parameters are chosen via cross validation and grid search is used to finalize the best parameters. We can see that our method can still achieve the best performance with the help of the designed CNN model. We can see the work from Manzoni *et al.* [6] shows much lower performance than that reported in their paper. This is because their data collection method is not specified in the paper and their method only uses the frequency features rather than combining time and frequency features in other works, which definitely has worse performance than the works combining time and frequency features shown in Table IV. Thus it performs worst among all methods.

TABLE V
SUMMARY OF PAST WORKS USING ACCELERATION TO DETECT TRANSPORTATION MODES

Study	Classes	Data	Window	Method	Accuracy
Hemminki *et al.* [3]	1.stationary 2.walk 3.bus 4.train 5.metro 6.tram	Accelerometer 60-100Hz pockets, bags	1.2 seconds	Adaptive boosting	80.1%
Manzoni *et al.* [6]	1.walk 2.bicycle 3.bus 4.car 5.metro 6.train 7.still 8.motorcycle	Accelerometer 25Hz	10.24 seconds	Decision tree	82.14%
Yang [29]	1.stand 2.sit 3.walk 4.run 5.bicycle 6.car	Accelerometer 36Hz	10 seconds	Decision tree	90.6%
This paper	1.stationary 2.walk 3.bicycle 4.bus 5.car 6.subway 7.train	Accelerometer 50Hz	10.24 seconds	Convolutional neural network	94.48%

TABLE VI
CLASSIFICATION RESULTS COMPARING WITH RELATED WORKS ON OUR DATASET

Algorithm	Window size		
	128	256	512
Hemminki *et al.* [3]	65.57%	72.38%	82.26%
Manzoni *et al.* [6]	63.38%	65.07%	67.31%
Yang [29]	76.42%	81.02%	88.07%
This paper	75.48%	82.42%	**94.48%**

Recently deep learning is employed in the transportation field [30]. Specifically, some studies employ deep learning to detect transportation modes [8]–[10], [31]. All of these studies use the data from GPS or other sensors, which are not comparable to our study in terms of energy consumption. In addition, Wang *et al.* [31] (74.1% in unknown number of transportation modes) and Fang *et al.* [8] (95.43% in five modes) use the deep neural network with fully-connected layers to perform the detection while Vu *et al.* [9] (93.1% in five modes) and Song *et al.* [10] (83.26% in five modes) employ recurrent neural networks to discover the relation among close samples. These works use the information from other sensors, such as GPS and Gyro, which is different from our work only using the accelerometer. None of them considers to apply CNNs in detecting transportation modes. In these works, all merge the motorised transportation modes into one cluster. If we also conduct the merge, the accuracy of our system can achieve 98%. In this paper, we show the CNNs can be successfully applied in transportation mode detection to achieve a high detection accuracy in detecting seven transportation modes.

Some other related work may use other sensors and extra information to detect transportation modes. A model based on sensor data from accelerometer, gyroscope, magnetic field, rotation vector, geomagnetic rotation vector, linear acceleration, and uncalibrated versions where applicable, is proposed in [32] to detect six transportation modes, including walk, bike, MRT, bus, car and stationary. The model employs four separate machine learning methods, gaussian naive bayes, discriminant analysis, SVM and k-NN. It can achieve 96% accuracy on average. Su *et al.* [33] propose an online SVM model to detect six transportation modes on the data from all motion sensors, including accelerometer, gravity sensor, gyroscope, magnetometer, and barometer. They can achieve an accuracy of 97.1%. Reddy *et al.* [18] propose a fusion model with decision trees and Hidden Markov Model (HMM) using GPS and accelerometer data to classify stationary, walk, run, bicycle and motorized transportation. They achieve an accuracy of 93.6%. Feng *et al.* [34] discover that combining

GPS and accelerometer can achieve higher accuracy than using GPS or accelerometer data only, and using accelerometer only has a higher accuracy than using GPS only in their model. Stenneth *et al.* [5] build a random forest model combining GPS and Geographical Information System (GIS) to classify stationary, walk, bicycle, car, bus, and train. They achieve an accuracy of 93.5%. Note that our system can achieve an accuracy of 94.48% using only accelerometer data.

Some other works detect users' activities using extra accelerometers or mobile devices. The work in [12] fuses the data from 5 biaxial accelerometers fixed at 5 body parts to recognize users' activities, such as walking, sitting, standing and running. A similar work [35] compares the performance of the acceleration data from 6 body parts in recognizing standing, sitting, walking and so on. Another work [36] presents a system to recognize the sitting, standing, lying and walking by requiring a device fixed at users' waists. But the above works require extra accelerometers. Kwapisz *et al.* propose a system in Android phones put in the front pants leg pocket to recognize users' 6 activities, such as sitting and walking [1]. Lee *et al.* use an HMM model to classify the activities [37] while Anguita *et al.* use a multiclass hardware-friendly SVM [2]. A fusion system of motion sensors is proposed to recognize physical activities in [38]. However, none of the above work focuses on detecting users' transportation modes using deep learning models.

VII. Conclusion

In this paper, we propose a robust system on Android smartphones to accurately detect users' transportation modes by employing the smartphone's accelerometer. To the best of our knowledge, this is the first system that utilizes convolutional neural networks to detect transportation modes with the accelerometer only. In this system, the collected data are processed by removing gravity and smoothing. The acceleration magnitude is used to build a convolutional neural network to recognize the corresponding transportation mode. Extensive experiments verify that the proposed system outperforms the CNNs of other architectures and traditional machine learning models. Our system can achieve an accuracy as high as 94.48%, which also outperforms the existing studies. Followup research aims to recognize more activities and build a more robust architecture by considering the context-aware information.

Appendix A
Background on Convolution, Max-Pooling, and Full-Connection in Deep Learning

The convolutional operation makes the presence of a pattern more important than the pattern's position. It is a basic module between two neural layers in a CNN. Units in a convolutional layer are organized by feature maps, which aggregates local patches in the feature map of the previous layer through a set of weights. The set of weights is called a filter bank. All units in a feature map share the same weights in a filter bank. In a feature map, the filter bank shifts a fixed length of step defined by the stride, to generate one unit. Different feature maps have different filter banks. Fig. 9 shows an example on how the calculation takes place in a convolutional layer. In the figure, the feature map from the previous layer is denoted by X, which is assumed to be a one-dimensional vector. The collection of filter banks is denoted by W, where the index j indicates the j^{th} filter bank. Every filter bank generates one feature map of the next layer by shifting a window on the previous layer's feature map. The shifting stride is predefined, which splits X into multiple segments, X_1, X_2,..., X_i, with the same size as the filter bank. The feature map of the next layer from the j^{th} filter bank is denoted by O_j, which contains the sum of multiplication between all segments in X and the filter bank. The value o_i of the i^{th} unit in the feature map is the sum of values multiplied by the weights in the filter,

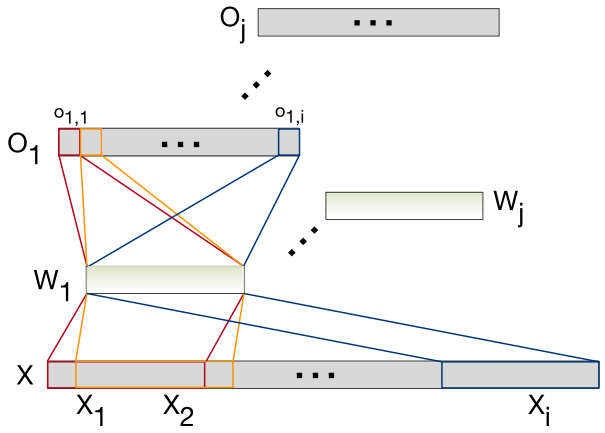


Fig. 9. The visualization of the calculation in a convolutional layer. The solid lines show how to generate the first feature map by the first filter bank W_1 and the lines of different colors correspond to different segments from the previous layer. The dotted lines show the generation of the second feature map.

$$o_{j,i} = \sum_k w_{j,k} x_{i,k} + b_j. \tag{13}$$

In the equation, $x_{i,k}$ and $w_{j,k}$ are elements in X_i and W_j, respectively. b_j is the bias for the unit. The value in a unit is the sum of all weighted values in a corresponding segment. The generated feature maps are used to generate the next layer's feature map. Thus, in a convolutional layer, three parameters need to be defined: the number of filter banks, the shape of a filter bank, and the shape of the stride.

The max pooling layer is used to pick out the salient values from a local patch of units. It can reduce the dimensionality by removing less important information in the feature map. Specifically, the max pooling operation selects the max value in a local patch of units and then the local patch shifts a step with the stride size. Thus, it is required to define the size of local patches and the stride's size in a max pooling layer.

The fully-connected layer is to connect all units in the previous layer to all units in the next layer. In a fully-connected layer, the number of units in the next layer is required to be set as a hyperparameter.

TABLE VII
FEATURES IN TRADITIONAL METHODS

Domain	Features
Time	mean, standard deviation, median, root mean square, min, max, range, kurtosis, and skewness
Frequency	log-scale power spectral density at 1-10Hz, and power spectral centroid

APPENDIX B
TRADITIONAL MACHINE LEARNING METHODS

We take the traditional machine learning methods, including Bayes Classifiers, C4.5 Decision Tree, K-Nearest Neighbors, Random Forest, Adaptive Boosting, Neural Network and Support Vector Machine, as the benchmarks for comparison [39]. In traditional machine learning methods, common features adopted in previous works [3], [4], [39] are selected from the training data to develop the models. The trained models are subsequently used to classify new data. In the following, the features and models in the traditional machine learning methods are explained briefly.

A. Features

In every window, the features are selected in both time and frequency domains, which are listed in Table VII, which are usually widely employed as features in peer works. When obtaining the features in the frequency domain, a window function is applied to the data.

The data in a window are scaled by a window function before they are transferred into the frequency domain. Window functions are shown effective in reducing the lobeside effect [40]. A commonly-used window function, Hamming, is employed in the system with the following formula,

$$w_n = \beta - \gamma \cos(\frac{2\pi n}{N}), \tag{14}$$

where w_n is the weight value in the window, β and γ are two constants and n is an integer from 1 to N. N is the number of samples in a window. By default, β is set to 0.54 and γ is set to 0.46, which are used to balance the information loss and reduce lobeside effect.

Till now, features in the traditional methods are obtained, but different features have different scales [25]. The normalization is applied on every feature in the data, which is the same process as shown in Section IV-.5.

B. Traditional Classification Models

Several common traditional machine learning models are adopted to classify the data. Brief introduction to every model is given in the following paragraphs. Every model's parameters are finely tuned by 10-folder cross validation via grid search. The parameters that perform best on the validation dataset are chosen to evaluate on the test dataset.

1) Bayes Classifiers: Bayes classifiers are statistical classifiers [41]. They predict an instance's class by calculating the probability that the instance belongs to each particular class via the similarity of feature values. The simplest one in Bayes classifiers is the Naive Bayes (NB), which assumes that all features are uncorrelated [41]. It calculates the probability of one instance X in one specific class C based on the Bayes' Theorem,

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}. \tag{15}$$

In the equation, $P(X)$ and $P(C)$ are known priors. In the assumption, the features are mutually independent, so $P(X|C)$ is the product of the probability of all features in one specific class. The instance is classified into the class with the maximum probability. A more complicated Bayesian classifier is Bayesian Networks (BNs) [42]. A Bayesian Network is a directed acyclic graph, which is to build a graph by the estimated correlation between features.

2) Decision Tree: The Decision Tree (DT) is to build a classification tree. The tree structure is presented as a leaf indicating a class and each node specifying some test on a single feature value with the branch and subtree for the possible outcome. To classify a case, it starts at the root and moves through the tree until a leaf is encountered [41]. The tree is split by information gain [43] and Gini index [44]. Only one feature is used to split the tree at every node. One of the most popular decision trees is C4.5 [41], which is used in this paper for comparison.

3) K-Nearest Neighbors: The K-Nearest Neighbors (K-NN) classifier is to classify an instance based on the closest k nearest neighbors in the training data [45]. It is called the lazy-learning algorithm, since its computation/overhead is much lighter during learning than testing time. The closeness between instances is defined as the distance, which is usually Euclidean or Manhattan distance. Among the k nearest neighbors, the instance is classified into the most common class, which class that the most neighbors belong to [41], [42]. The performance of K-NN may be affected by the choice of k.

4) Random Forest: The Random Forest (RF) method uses ensembles of unpruned decision trees [46], [47]. A common decision tree is usually pruned to avoid overfitting, but in random forest, the decision trees are unpruned. It draws bootstrap samples from the training data. It randomly chooses a subset of features in the samples to build a complete decision tree according to the samples. Multiple decision trees are built with different samples in the same way. The classification result is predicted by aggregating the classification results from all trees.

5) Adaptive Boosting: The Adaptive Boosting (AB) is to train multiple weak classifiers from subsets with the same size. The final classification result is obtained by aggregating the classifiers with weights. The weights are adaptive. If one instance outside the subset in the training data is classified correctly, then the weight is reduced; otherwise it increases [48].

6) Neural Network: The Neural Network (NN) is a set of connected input/output units in which a weight is associated with each connection. A Neural network is usually composed

of an input layer, one or more hidden layers and an output layer. The data is received in the input layer and processed in the hidden layers. The output layer produces the classification results [41]. The network is built by updating weights via backpropagation.

7) Support Vector Machine: The Support Vector Machine (SVM) builds a hyperplane to separate two data classes by maximizing the margin between two classes and the hyperplane based on a cost function. The SVM is at first outlined for linearly separable cases. A kernel function is defined to transfer nonlinear features into linear ones with high dimensions [41], [49]. The SVM classifies multiple classes via training several SVMs on every two classes, or every one class and another class including all the data in the other classes. In this experiment, we use the "radial basis function" as the kernel function. Gamma is 10 over the number of features, and the decision function is "one-vs-one".

REFERENCES

[1] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SIGKDD Explor. Newslett.*, vol. 12, no. 2, pp. 74–82, May 2011.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Ambient Assisted Living and Home Care*. Vitoria-Gasteiz, Spain: Springer, Dec. 2012, pp. 216–223.

[3] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proc. 11th ACM Int. Conf. Embedded Netw. Sensor Syst.*, Roma, Italy, Nov. 2013, pp. 13:1–13:14.

[4] A. Jahangiri and H. A. Rakha, "Applying machine learning techniques to transportation mode recognition using mobile phone sensor data," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2406–2417, Oct. 2015.

[5] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, "Transportation mode detection using mobile phones and GIS information," in *Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Chicago, IL, USA, Nov. 2011, pp. 54–63.

[6] V. Manzoni, D. Maniloff, K. Kloeckl, and C. Ratti, "Transportation mode identification and real-time Co_2 emission estimation using smartphones," SENSEable City Lab, Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2010.

[7] P. Widhalm, P. Nitsche, and N. Brändie, "Transport mode detection with realistic smartphone sensor data," in *Proc. 21st Int. Conf. Pattern Recognit. (ICPR)*, Istanbul, Turkey, Nov. 2012, pp. 573–576.

[8] S.-H. Fang, Y.-X. Fei, Z. Xu, and Y. Tsao, "Learning transportation modes from smartphone sensors based on deep neural network," *IEEE Sensors J.*, vol. 17, no. 18, pp. 6111–6118, Sep. 2017.

[9] T. H. Vu, D. Le, and J. C. Wang, "Transportation mode detection on mobile devices using recurrent nets," in *Proc. 24th ACM Int. Conf. Multimedia*, Amsterdam, The Netherlands, Oct. 2016, pp. 392–396.

[10] X. Song, H. Kanasugi, and R. Shibasaki, "Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, New York, NY, USA, Jul. 2016, pp. 2618–2624.

[11] Google-Android. (2017). *Motion Sensors*. Accessed: Aug. 2017. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_motion.html

[12] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," in *Pervasive Computing*. Tokyo, Japan: Springer, Sep. 2004, pp. 1–17.

[13] B. Nham, K. Siangliulue, and S. Yeung, "Predicting mode of transport from iPhone accelerometer data," Mach. Learn. Final Projects, Stanford Univ., Stanford, CA, USA, Tech. Rep., Dec. 2008.

[14] Google-Android. (2017). *Sensor Event*. Accessed: Aug. 2017. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorEvent.html

[15] J. Karki, "Active low-pass filter design," Texas Instrum., Dallas, TX, USA, Appl. Rep. SLOA049B, Sep. 2000.

[16] Google-Android. (2017). *Android Accelerometer*. Accessed: Aug. 2017. [Online]. Available: http://www.kircherelectronics.com/blog/index.php/11-android/sensors/10-low-pass-filter-linear-acceleration

[17] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Anal. Chem.*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964.

[18] S. Reddym, M. Mun, J. Burke, D. Estrin, M. Hansen, and Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sensor Netw.*, vol. 6, no. 2, pp. 13-1–13-27, Feb. 2010.

[19] I. J. Good, "The interaction algorithm and practical Fourier analysis," *J. Roy. Stat. Soc., B (Methodol.)*, vol. 20, no. 2, pp. 361–372, Jan. 1958.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Stateline, NV, USA, Dec. 2012, pp. 1097–1105.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. Int. Conf. Comput. Vis.*, Santiago, Chile, Dec. 2015, pp. 1026–1034.

[22] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Proc. Adv. Neural Inf. Process. Syst.*, Stateline, NV, USA, Dec. 2013, pp. 351–359.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[24] S. Ruder, "An overview of gradient descent optimization algorithms," Sep. 2016, *arXiv:1609.04747*. [Online]. Available: https://arxiv.org/abs/1609.04747

[25] X. Liang, J. Tian, X. Ding, and G. Wang, "A risk and similarity aware application recommender system," *J. Comput. Inf. Technol.*, vol. 23, no. 4, pp. 303–315, 2015.

[26] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," Mar. 2016, *arXiv:1603.04467*. [Online]. Available: https://arxiv.org/abs/1603.04467

[27] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] J. Yang, "Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones," in *Proc. 1st ACM Int. Workshop Interact. Multimedia Consum. Electron.*, Beijing, China, Oct. 2009, pp. 1–10.

[30] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1243–1253, Feb. 2019.

[31] H. Wang, G. Liu, J. Duan, and L. Zhang, "Detecting transportation modes using deep neural network," *IEICE Trans. Inf. Syst.*, vol. 100, no. 5, pp. 1132–1135, May 2017.

[32] M. Kodyš, P. Oliver, J. Bellmunt, and M. Mokhtari, "Human urban mobility classification in AAL deployments using mobile devices," in *Proc. 19th Int. Conf. Inf. Integr. Web-Based Appl. Services*, Salzburg, Austria, Nov. 2017, pp. 311–319.

[33] X. Su, H. Caceres, H. Tong, and Q. He, "Online travel mode identification using smartphones with battery saving considerations," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 10, pp. 2921–2934, Oct. 2016.

[34] T. Feng and H. J. P. Timmermans, "Transportation mode recognition using GPS and accelerometer data," *Transp. Res. C, Emerg. Technol.*, vol. 37, pp. 118–130, Dec. 2013.

[35] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher, "Activity recognition and monitoring using multiple sensors on different body positions," in *Proc. Int. Workshop Wearable Implantable Body Sensor Netw. (BSN)*, Cambridge, MA, USA, Apr. 2006, p. 4 and 116.

[36] F. R. Allen, E. Ambikairajah, N. H. Lovell, and B. G. Celler, "An adapted Gaussian mixture model approach to accelerometry-based movement classification using time-domain features," in *Proc. Int. Conf. IEEE Eng. Med. Biol. Soc.*, New York, NY, USA, Aug./Sep. 2006, pp. 3600–3603.

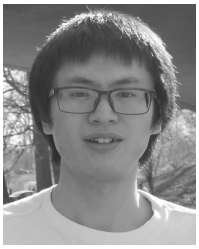
[37] Y.-S. Lee and S.-B. Cho, "Activity recognition using hierarchical hidden Markov models on a smartphone with 3d accelerometer," in *Hybrid Artificial Intelligent Systems*. Wrocław, Poland: Springer, May 2011, pp. 460–467.

[38] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. M. Havinga, "Fusion of smartphone motion sensors for physical activity recognition," *Sensors*, vol. 14, no. 6, pp. 10146–10176, Jun. 2014.

[39] X. Liang and G. Wang, "A convolutional neural network for transportation mode detection based on smartphone platform," in *Proc. IEEE Int. Conf. Mobile Ad Hoc Sensor Syst.*, Oct. 2017, pp. 338–342.

[40] A. H. Nuttall, "Some windows with very good sidelobe behavior," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 1, pp. 84–91, Feb. 1981.

[41] H. Bhavsar and A. Ganatra, "A comparative study of training algorithms for supervised machine learning," *Int. J. Soft Comput. Eng.*, vol. 2, no. 4, pp. 74–81, Sep. 2012.
[42] T. N. Phyu, "Survey of classification techniques in data mining," in *Proc. Int. MultiConf. Eng. Comput. Sci.*, Hong Kong, vol. 1, Mar. 2009, pp. 1–5.
[43] R. A. Hunt, "On L(p, q) spaces," *Enseign. Math*, vol. 12, no. 2, pp. 249–276, 1966.
[44] R. I. Lerman and S. Yitzhaki, "A note on the calculation and interpretation of the Gini index," *Econ. Lett.*, vol. 15, nos. 3–4, pp. 363–368, 1984.
[45] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[46] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, Aug. 1995, pp. 278–282.
[47] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, "Random forest: A classification and regression tool for compound classification and QSAR modeling," *J. Chem. Inf. Comput. Sci.*, vol. 43, no. 6, pp. 1947–1958, Nov. 2003.
[48] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[49] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.



Xiaoyuan Liang received the B.S. degree in computer science and technology from the Harbin Institute of Technology, China, in June 2013, and the Ph.D. degree from the Computer Science Department, New Jersey Institute of Technology, in August 2019. He is currently a Research Scientist at Facebook. His research interests include deep learning, data mining, and vehicular networks.

Yuchuan Zhang is currently pursuing the master's degree from NJIT. He currently works on Microstrategy as a Software Engineer Big Data Engine Team. He works on data preparation recommendation and data analysis recommendation. Previously, he researched on mobile computing, smart transportation, and data mining at internet laboratory of NJIT.

Guiling (Grace) Wang received the B.S. degree in software from Nankai University, Tianjin, China, and the Ph.D. degree in computer science and engineering and a minor in statistics from The Pennsylvania State University in May 2006. She is currently a Professor with the Yingwu College of Computing Sciences. She also holds a joint appointment at the MT School of Management. In July 2006, she joined NJIT as an Assistant Professor and was promoted to an Associate Professor with tenure in June 2011. She was promoted to a Full Professor in June 2016 in her 30s. Her research interests include deep learning applications, blockchain technologies, intelligent transportation, and mobile computing.

Songhua Xu received the M.S., M.Phil., and Ph.D. degrees in computer science from Yale University. He is currently an Assistant Professor with the Information Systems Department, New Jersey Institute of Technology. His primary research interests include information retrieval and management, web search and data mining, innovative applications of AI, and intelligent systems for biomedical applications. He also has secondary research interests spanning across visual computing, multimedia, human–computer interaction, computer graphics, visualization, as well as digital arts and design. He is particularly passionate about building human-centered applications that benefit people and society through advanced computing techniques.