Transportation Research Part C 103 (2019) 142–157

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

DeepTrend 2.0: A light-weighted multi-scale traffic prediction model using detrending

Xingyuan Dai[a,b,d], Rui Fu[c], Enmin Zhao[c], Zuo Zhang[c], Yilun Lin[a,b,d], Fei-Yue Wang[a,b,d], Li Li[c,*]

[a] The State Key Laboratory for Management and Control of Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[b] University of Chinese Academy of Sciences, Beijing 100049, China
[c] Department of Automation, Tsinghua University, Beijing 100084, China
[d] Qingdao Academy of Intelligent Industries, Qingdao, Shandong 266109, China

ARTICLE INFO

Keywords:
Traffic prediction
Deep learning
Detrending
Multi-scale traffic prediction

ABSTRACT

In this paper, we propose a detrending based and deep learning based many-to-many traffic prediction model called DeepTrend 2.0 that accepts information collected from multiple sensors as input and simultaneously generates the prediction for all the sensors as output. First, we demonstrate that detrending brings advantages to traffic prediction, even when deep learning models are considered. Second, the proposed model strikes a delicate balance between model complexity and accuracy. In contrast to the existing models that view a sensor network as a weighted graph and use graph convolutional neural networks (GCNN) to model spatial dependency, we represent a sensor network as an image and propose a convolutional neural network (CNN) as the prediction model. The image is generated by the correlation coefficient between the flow series of sensors, which is different from other CNN based prediction approaches that convert the transportation network into an image by the spatial location of sensors or regions. Compared with the GCNN based model, the CNN based DeepTrend 2.0 can achieve much faster convergence during training, and it guarantees similar prediction quality. Test results indicate that the proposed light-weighted model is efficient and easy to transfer and deploy.

1. Introduction

Traffic flow prediction has attracted sustained interest during the last four decades; however, it has not been completely solved. It deals with predictions for future traffic flow based on limited historical traffic data. The original research mostly applied a classical statistical model to make predictions for a single point (Vlahogianni et al., 2014). Subsequently, with the growth in traffic data, a variety of data-driven methods (including some machine learning models) were proposed. These approaches are capable of mining complex features of traffic flow and achieve better performance, especially for complex traffic conditions (Vlahogianni et al., 2014). In the preceding four years, the deep learning models became the focal point in this field. Deep architecture can extract the high-level feature of traffic flow and obtain state-of-the-art prediction performance.

In this study, we focus our attention on many-to-many prediction. Many-to-many prediction is necessary for intelligent transportation systems (ITS) because it involves massive sensing units, and considerable time and resources would be consumed if we built

* Corresponding author at: Room 806, Central Main Building, Tsinghua University, Beijing 100084, China.
E-mail address: li-li@tsinghua.edu.cn (L. Li).

https://doi.org/10.1016/j.trc.2019.03.022
Received 11 May 2018; Received in revised form 2 March 2019; Accepted 23 March 2019
0968-090X/ © 2019 Elsevier Ltd. All rights reserved.

a prediction model for each sensor station. Many-to-many prediction embeds all sensor stations in a model; this improves the modeling efficiency and can further facilitate the prediction ability by capturing spatial dependency of the sensor network in the model.

In this paper, we address two of the remaining problems of many-to-many prediction that were raised by Vlahogianni et al. (2014).

First, whether detrending is still necessary when a deep learning model is introduced needs to be elucidated. Most existing deep learning based prediction models use only original traffic flow to make predictions without considering trend information. However, we usually consider limited historical data (e.g., traffic flow data observed in the last half hour) as the input of the prediction model to obtain prediction output. As a result, the prediction model may not be fully aware of the trend of traffic flow that appeared beyond such a limited time horizon (time window).

A simple solution to handle this problem is to apply detrending, i.e., we use the residual time series rather than the original traffic flow to make predictions. There are various detrending methods, e.g., one typical approach (Okutani and Stephanedes, 1984; Williams and Hoel, 2003) adopts a one-week lagged first seasonal difference of traffic flow series for prediction resting on the assertion that weekly seasonal difference will yield a stationary transformation. Another approach is to retrieve the intra-day trend that describes the typical flow pattern shared by consecutive days, and is defined by daily/weekly-average trend (Chen et al., 2012; Li et al., 2014), time-variant trend (Cetin and Comert, 2006; Kamarianakis et al., 2012), principal component analysis (Qu et al., 2009), or wavelet-based decomposition (Boto-Giralda et al., 2010). This approach further builds a prediction model based on the detrended time series. These methods embed the traffic variation pattern into the residual time series to forecast traffic flow; experimental results indicate that the prediction performance may improve significantly if detrending is used. However, few deep learning based prediction models have applied detrending.

Second, how to balance the performance and complexity of the prediction model needs to be carefully considered. Although the amount of data has increased dramatically over the years owing to the rapidly growing number of sensing units in the ITS, we usually do not need most of the data to build a particular prediction model (Li et al., 2015a). Several papers considered how to filter out the sensors that are not causally related (Cheng et al., 2017; Li et al., 2017b). Moreover, we also aim to find a light-weighted deep learning prediction model that is easy to deploy, because the computational costs consumed by prediction are essential in practice.

To answer both the abovementioned problems, we propose a traffic prediction model using detrending called DeepTrend 2.0. Our main contributions can be summarized as follows:

First, we demonstrate that detrending is useful for deep learning models to forecast traffic flow. We use the simple average trend method to model the periodic pattern in traffic flow for all the sensors and build a forecasting model for the detrended time series obtained by eliminating the trend time series in the original time series. The detrending scheme that decouples the trend model and many-to-many prediction model enables the prediction model to capture the spatial-temporal dependency of traffic variation pattern on a large scale. Experimental results indicate that detrending will enhance the performance of the deep learning based many-to-many prediction.

Second, we balance the performance and complexity of the prediction model. If we feed relatively long-range historical data (e.g., traffic flow data observed over the past few days) as the input of the deep neural network prediction model, we may obtain better prediction results because we could naturally consider the trend of traffic flow that appeared after such a limited time horizon (time window). However, this significantly increases the training and implementation costs of the model. In our previous study: DeepTrend 1.0 (Dai et al., 2017), we demonstrated that putting original historical data directly into a deep neural network may not be a good choice.

The proposed model: DeepTrend 2.0 is different from existing many-to-many prediction models in the regard that it views a sensor network as a weighted graph and uses graph convolutional neural networks (GCNN). Our model converts a sensor network into an image and employs the convolutional neural network (CNN). Empirical results indicate that CNN is more efficient and attains similar prediction quality as GCNN. It should be noted that the formation approach of the image in this study varies from the following two principal approaches: in the approach proposed in Zhang et al. (2017a), spatial information is formalized as an image, where the location of the pixel points reflects the spatial location of different areas of the transportation network, whereas temporal information is reflected in different channels of the image. In another approach proposed in Wu et al. (2018), the linear spatial relation of sensors along the highway corridor and the temporal relation are embodied in two dimensions of a one-channel image. In the proposed formalization method, the image is obtained by the correlation between sensors; in the image, the pixel points representing two sensors with a strong correlation are neighbors, and different channels indicate the temporal information. Therefore, this formalization method guarantees the local similarity of the image. DeepTrend 2.0 enables us to take advantage of the high representational capability of CNN to mine the spatial-temporal feature in traffic flow at the cost of a small number of computational resources.

To explain our findings further, the rest of this paper is organized as follows. Section 2 briefly reviews the existing deep learning based traffic flow prediction models proposed in the literature. Section 3 first introduces detrending based prediction and then presents the hierarchical deep neural network model: DeepTrend 2.0. Section 4 compares the original data based prediction with the detrending based prediction and the GCNN prediction model with DeepTrend 2.0. Finally, Section 5 concludes the paper.

2. Literature review

The core aim of traffic flow prediction is to establish a particular model appropriately that accepts historical traffic flow data as input and yields future traffic flow data as output. Various prediction models such as autoregressive moving average (ARMA) (Ahmed

and Cook, 1979; Levin and Tsao, 1980), multi-variable linear regression (MVLR) (Li et al., 2015b), supporting vector regression (SVR) (Castro-Neto et al., 2009; Jin et al., 2007), and random forests (Leshem and Ritov, 2007) have been proposed to reach this goal.

In the last five years, several researchers began applying deep learning techniques to build traffic flow prediction models (Cao et al., 2017; Fu et al., 2016; Huang et al., 2014; Koesdwiady et al., 2016; Li et al., 2017a; Lin et al., 2017, 2018; Lv et al., 2014; Ma et al., 2017, 2015a, 2015b; Polson and Sokolov, 2017; Tan et al., 2016; Yu et al., 2017a, 2017b; Zhang et al., 2017b; Zhao et al., 2017) because deep learning technique gained remarkable achievements in several other fields. In this study, we focus on deep learning based traffic prediction approaches.

According to the input and output dimensions of prediction models, deep learning models for traffic time series prediction can be divided into three categories: one-to-one prediction, many-to-one prediction, and many-to-many prediction.

A one-to-one prediction model uses only historical traffic information of one sensor to predict its future traffic condition (e.g., travel flow, travel time, and travel speed). In our previous model: DeepTrend 1.0 (Dai et al., 2017), we used an extraction layer to obtain time-variant trend of traffic flow for improved prediction for a sensor. Ma et al. (2015a) employed long short-term memory (LSTM) network to capture the non-linear relationship between the future traffic condition and past traffic information of one sensor. Yu et al. (2017c) leveraged LSTM with multiple hidden layers to identify the hierarchical temporal features of historical traffic flow and forecast peak-hour traffic flow. Additionally, Fu et al. (2016) were the first to apply gated recurrent units (GRU) network to one-to-one traffic flow prediction.

In contrast, a many-to-one prediction model uses historical information of multiple sensors to predict the traffic condition of one sensor. For example, a deep architecture was introduced to perform many-to-one prediction in Huang et al. (2014). In this architecture, a deep belief network (DBN) was used for unsupervised feature learning of the historical flow in all observation sensors, and a regression layer above the DBN was used for traffic flow prediction on a single sensor. However, this model did not fully utilize spatial information to build prediction models. Unrelated sensors may also be considered for prediction; however, they can introduce errors in the prediction. DBN is insufficient to deal with complex spatial correlation; therefore, the prior spatial knowledge was considered in Tan et al. (2016). In this architecture, the input of the DBN model only included correlated information from the upstream and downstream of the target sensor. However, in this model, training was performed for each target sensor separately; therefore, it consumed several computation resources.

On the contrary, many-to-many prediction uses historical information of multiple sensors to predict the traffic conditions of multiple sensors simultaneously. The aforementioned deep architecture (Huang et al., 2014) can be extended to this type of prediction task by applying a multi-output regression layer on top of the DBN to integrate several prediction tasks. Another representative study (Lv et al., 2014) used stacked auto-encoders (SAE) to extract the generic flow feature and a multi-output regression layer above the SAE to predict the traffic flow of multiple sensors. In recent studies, more prior spatial relationship information such as the spatial position and correlation coefficient of the sensors is considered. This promotes the deep network to capture the spatial dependency in an improved manner. Another study proposed in Zhao et al. (2017) used a matrix to maintain the prior spatial-temporal correlation information and encoded the traffic flow in multiple sensors into feature representation, which was further used as the input of the memory units of LSTM to predict the flow of all sensors in the next time interval.

To make improved use of the prior spatial relationship information to mine the spatial feature of the traffic network, some studies use convolutional neural networks (CNNs) (Lecun et al., 1998) for many-to-many prediction. For example, a study proposed in Cao et al. (2017) first represented the communication traffic between different services of an entire network at a particular time as an image frame of the video. Further, they used CNN to capture the interactive features among all the services pairs in the image and finally used GRU to predict the temporal dependency between the previous and future image frames of the service traffic. Yu et al. (2017b) proposed a grid-based network-segmentation method to view transportation network as an image, in which the pixel value represents the traffic state in the network. Further, they employed CNN to mine the near and distance spatial dependency and utilized LSTM for multi-step predictions. Another study (Zhang et al., 2017a, 2017b) partitioned a city into a grid map, wherein the value of the grid denotes the crowd flow in the corresponding region. The historical flows categorized into recent, near, and distant temporal properties were separately fed into a ResNet (He et al., 2016), which is a special CNN structure, to obtain three spatial-temporal features that were finally aggregated to forecast the future crowd flows for all the regions in the map. The aforementioned approaches view network evolution as a multi-channel image, wherein each channel only reflects the spatial information, and different channels reflect the temporal information. However, some studies (Ma et al., 2017; Dixon et al., 2018; Kim et al., 2018; Yang et al., 2018) used the time-space image of a channel to embody the time and space dimension information of the transportation network. The differences between them are that Ma et al. (2017) used a CNN model, Dixon et al. (2018) used a recurrent neural network (RNN) model, and Kim et al. (2018) applied neural network with capsules. Moreover, Yang et al. (2018) first identified critical road sections in the image as the input and used a hybrid convolutional long short-term memory neural network (ConvLSTM) to predict the future traffic states of the entire transportation network. Some other studies (Cui et al., 2018; Li et al., 2017a; Lin et al., 2017; Yu et al., 2017a) treated traffic network as a graph in which the nodes represented sensor stations and contained traffic information, and the entries of the adjacency matrix quantify the associations between sensors. Further, these approaches employed GCNN (Defferrard et al., 2016; Kipf and Welling, 2017) to mine the spatial feature, which was fed into the temporal model to perform spatial-temporal traffic series forecasting. However, few studies compared the performance of different methods to utilize spatial patterns from different sensors.

3. Methodology

3.1. Detrending based prediction

As illustrated in Li et al. (2014), detrending based predictions mainly include three steps: (1) calculating a trend, (2) separating the residual (detrended) time series from the trend, and (3) building a prediction model for the residual time series.

To calculate the trend, we use a method called simple average trend that uses the average of the periodic traffic flow series in each sensor as its traffic flow trend. Let $y_{wa,b}$ represent the bth flow data point in the ath week at a sensor. We denote the traffic flow series in M continuous weeks as a series of vectors

$$\mathbf{y}_{w1} = [y_{w1,1}, y_{w1,2}, \cdots, y_{w1,n_w}], \ \cdots, \mathbf{y}_{wM} = [y_{wM,1}, y_{wM,2}, \cdots, y_{wM,n_w}], \tag{1}$$

where n_w is the number of sample data in a week. If the sample time interval is five minutes, we have $n_w = 288 \times 7$. Then, the simple average trend over past M weeks is computed as

$$\mathbf{o} = [o_{w,1}, \cdots, o_{w,n_w}] = \left[\frac{1}{M}\sum_{a=1}^{M} y_{wa,1}, \cdots, \frac{1}{M}\sum_{a=1}^{M} y_{wa,n_w}\right], \tag{2}$$

where $o_{w,b}$ denotes the trend value of the bth flow data point in a week.

After obtaining the trend, the residual time series $\mathbf{r}_{wa} = [r_{wa,1}, r_{wa,2}, \cdots, r_{wa,n_w}]$ in the ath week is calculated by subtracting the intra-week trend from the original flow series as

$$\mathbf{r}_{wa} = \mathbf{y}_{wa} - \mathbf{o}. \tag{3}$$

For detrending based traffic flow prediction, the residual time series in N continuous time intervals, which is denoted as $[r_{t-(N-1)}, r_{t-(N-2)}, \cdots, r_t]$, where r_t represents the residual value at time t, is fed into the prediction model, and the predicted future residual value $\hat{r}_{t+1}$ at time $(t+1)$ is added to the corresponding trend value o_{t+1} to obtain the predicted flow value as $\hat{y}_{t+1} = \hat{r}_{t+1} + o_{t+1}$.

In the following subsections, we present the prediction model. This prediction model is decoupled from the trend model; therefore, the detrending based method and the original data based method that uses the original traffic flow for prediction can share the same architecture of the prediction model. We use the original data based method as an example to present the proposed prediction model. The detrending based method only needs to replace the original flow series with a residual time series.

3.2. Formulation of many-to-many traffic flow prediction problem

To model spatial dependency for many-to-many prediction, an intuitive approach is to partition the traffic network into a grid-based map (Yu et al., 2017b; Zhang et al., 2017a). The method is based on the hypothesis that there is a possible causal relationship between different areas in the map, especially for neighboring areas, and these causal sources of information will be beneficial to traffic prediction. However, in this study, we focus on the traffic flow in sensors that are unevenly distributed in the space. If each sensor occupies a pixel in the image by its spatial location, the obtained image-like matrix is large and sparse, which wastes computing resources and makes it hard to handle the spatial dependency.

It was recently shown in Li et al. (2017b) that a strongly correlated source of information often indicates a causal relationship in traffic flow prediction. Therefore, rather than using spatial locations of sensors, we use statistical correlations between them to model the spatial dependency and propose a heuristic method to map sensors to a two-dimensional image.

We adopt the Spearman rank correlation coefficient (Zwillinger and Kokoska, 1999) to approximately determine the causal relationship between sensors, which is calculated as follows:

$$r(s_k, s_l) = \frac{\sum_{t=1}^{T}(u_t^{s_k} - \bar{u}^{s_k})(u_t^{s_l} - \bar{u}^{s_l})}{\sqrt{\sum_{t=1}^{T}(u_t^{s_k} - \bar{u}^{s_k})^2 \sum_{t=1}^{T}(u_t^{s_l} - \bar{u}^{s_l})^2}}, \tag{4}$$

where u represents ranks of the flow series x, s_k and s_l are the sensor identifiers, $\bar{u}^{s_k}$ and $\bar{u}^{s_l}$ are means of ranks of two series in sensors s_k and s_l, and T is the series length. Spearman rank correlation coefficient accesses monotonic relationships between two variables and has a value between -1 and 1. If the absolute value of the coefficient is closer to 1, the two sensors will have stronger correlation. We define the correlation matrix as:

$$Corr(s_k, s_l) = |r(s_k, s_l)|, \tag{5}$$

and denote the dimensions of the correlation matrix as $n \times n$ (n also represents the number of sensors).

After obtaining the correlation matrix, we embody all the sensors in an image, in which each pixel represents a sensor identifier, and further make pixels of strongly correlated sensors to be neighbors because strongly correlated sensors are generally very close in space.

Considering that there are n sensors, we first map all the sensors to a square image I of $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ pixels in the order of sensor identifiers, where $\lceil\cdot\rceil$ is the ceiling function. The remaining $(\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil - n)$ pixels of the image that is not filled with sensors can be padded with zeros. Fig. 1(a) illustrates an image of 5×5 pixels that embodies 25 sensors. We define the α-neighbors of a pixel as the other elements that have α units distance from it in horizontal, vertical, and diagonal directions. Fig. 1(b) illustrates the 1-neighbors of the pixel; we can see that the inner pixel of the image such as sensor 7 has eight 1-neighbors, one at an edge such as sensor 6 has 5,

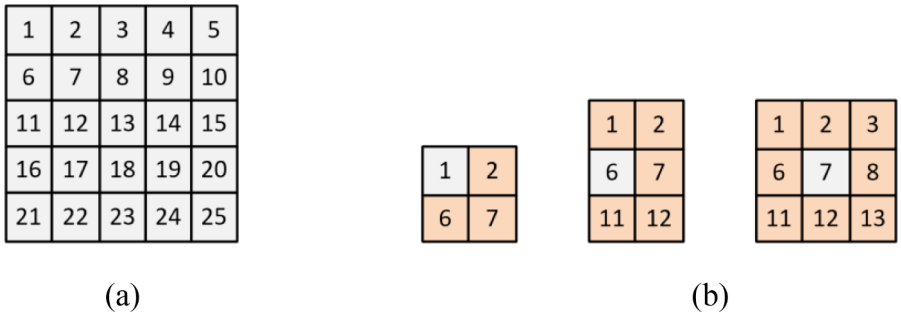

Fig. 1. Formalize traffic sensors as an image. (a) Each pixel represents a sensor identifier. (b) Pixel under consideration (grey) and its 1-neighbors (orange). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and one at a corner such as sensor 1 has 3.

We denote the pixel located at (i, j) of the image I as $I(i, j)$, and its α-neighbors as $E_{(i,j)} = \{e^1_{(i,j)}, e^2_{(i,j)}, \cdots, e^G_{(i,j)}\}$. We define the score of pixel $I(i, j)$ as

$$score(i, j) = \frac{1}{G}\sum_{g=1}^{G} Corr(I(i, j), e^g_{(i,j)}). \tag{6}$$

Then, the total score of the image is

$$Score = \sum_{i=1}^{\lceil\sqrt{(n)}\rceil}\sum_{j=1}^{\lceil\sqrt{(n)}\rceil} score(i, j). \tag{7}$$

To make the pixels of strongly correlated sensors be close, we rearrange the order of the sensors in the image to maximize the total score. However, this is an NP-hard problem, which is difficult to solve directly. Therefore, we consider solving the problem approximately by iteration. For a certain pixel, we look for its swap and other pixels that achieve the maximum total score after swapping the values of two pixels. We then make a swap. This process is performed for all the pixels and repeated until there is no swap that can further improve the total score. Finally, we can obtain a rearranged sensor image I_r. This process is summarized in Algorithm 1, and its complexity is $O(\mathrm{n}^2)$ because we have two for-loops that traverse n and $(n-1)$ elements separately.

Algorithm 1 (*Rearrange the order of sensors in the image*).

Input: Input image I.
Output: Image I_r after rearranging sensor order.
Compute the total score of the input image *old_score*.
Initialize a matrix: *new_score* of the same size as I to save the new total score while swapping two pixels.
while TRUE **do**
  swap_num = 0% Count the number of swaps in an iteration.
  for all pixels $I(i_1, j_1)$ **do**
    while TRUE **do**
      Reset all elements of *new_score* to be zero.
      for all pixels $I(i_2, j_2)$ excepting $I(i_1, j_1)$ **do**
        Swap $I(i_1, j_1)$ and $I(i_2, j_2)$.
        Compute the new total score and save it as $new_score(i_2, j_2)$.
        Swap $I(i_1, j_1)$ and $I(i_2, j_2)$. % Change pixels back to original positions.
      if $\max(new_score) - old_score > 10^{-5}$ **then**
        $(i_3, j_3) = arg\max(new_score)$
        Swap $I(i_1, j_1)$ and $I(i_3, j_3)$.
        $old_score = \max(new_score)$
        $swap_num = swap_num + 1$
      else
        break
  if $swap_num = 0$ **then**
      $I_r = I$
    return I_r

In this study, we map 380 sensors to an image of 20×20 pixels. Fig. 2 illustrates the correlation coefficient between the sensor with ID 401423 (red square) in the PeMS dataset and other sensors in original and rearranged order of the image. After rearranging the order, it can be seen that not only the sensor (ID 401423) and its neighboring sensors have strong correlations but other sensors

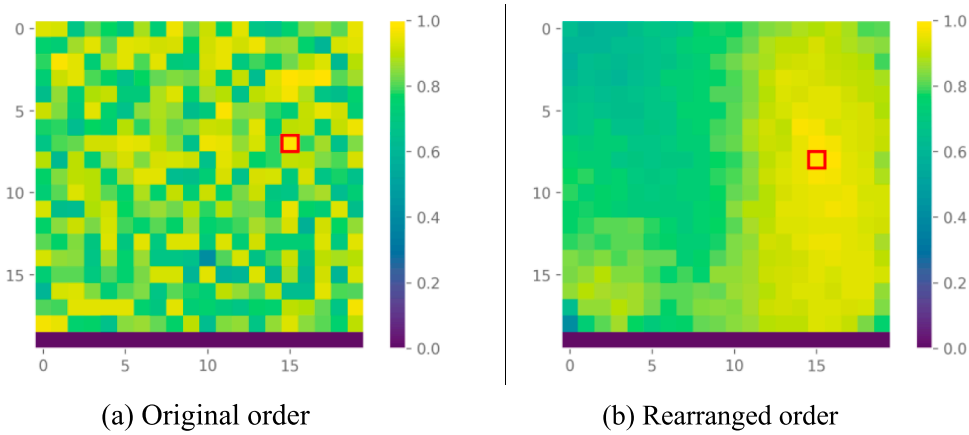

Fig. 2. Visualization of correlation coefficients between sensor with ID 401423 (indicated by red square) and other sensors in original order and rearranged order of image. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

with similar correlation coefficients at the sensor (ID 401423) are clustered together. This type of local similarity leads CNN to capture the spatial feature of traffic flow in an improved manner.

At the tth time interval, the traffic flow in all sensors can be denoted as a one-channel image X_t, where the pixel located at (i, j) of X_t represents the flow of sensor $I_r(i, j)$. Then, the traffic flow in all the sensors in N continuous time intervals can be turned into an N-channel image $\mathbf{X} = [X_{t-(N-1)}, X_{t-(N-2)}, \cdots, X_t]$. In this study, we consider multi-scale many-to-many prediction and use the historical flow of 5 min time intervals to predict the future traffic flow in 5, 15, 30, and 45 min simultaneously. We denote the time lag as N. The problem can be formalized as follows: *Given the historical traffic flow* $[X_{t-(N-1)}, X_{t-(N-2)}, \cdots, X_t]$, *predict* $[X_{t+1}, X_{t+3}, X_{t+6}, X_{t+9}]$.

3.3. CNN for capturing Spatial-Temporal features

The constructed image embeds the spatial-temporal dependency of the sensor network, where pixels representing the traffic volumes of strongly correlated sensors are neighboring, and traffic volumes of the same sensor in different time intervals are in different channels. We naturally utilize CNN with multiple convolutional layers to capture the spatial-temporal features of the traffic network. The convolutional layer has many convolutional kernels (or filters), all of which can extract different abstract features of the image. We denote the input of the lth convolutional layer as O^{l-1} and the output as O^l. The convolution process can be described as follows:

$$O_k^l = f\left(\sum_h W_{h,k}^l * O_h^{l-1} + b_k^l\right), \tag{8}$$

where $*$ denotes the convolution operation, f is an activation function, $W_{h,k}^l$ and b_k^l are the learnable parameters, h and k are the channel indices of the input and output, and k additionally indicates the index of the convolutional kernel.

Weights sharing in the convolutional layer may make it limited to distinguish local features of the traffic network. To better capture spatial features, we consider the locally connected layer (Pang et al., 2017). Like the convolutional layer, the locally connected layer is designed to exploit the local correlations of the image and process each locally connected patch of the image. The difference between them is that the locally connected layer does not share the weights when applied at each different patch of the image. Therefore, locally connected layers may have more capacity to model the spatial relationship. However, abundant training parameters in locally connected layers will introduce more complexity, and we also need convolutional layers to build a light-weighted model.

Since convolutional and locally connected layers have a similar operation to the input, we use convolutional layers to illustrate the process of extracting spatial-temporal features. A convolutional layer can only capture the local spatial dependency limited by its kernel size, and a stack of convolutional layers can account for larger and even global dependency. As shown in Fig. 3(a), there are three multiple levels of feature maps that are connected with convolutions with kernel size 3×3. We find that a point in the high-level feature map depends on 9 points in the middle-level feature map, and all of those points depend on 25 points in the low-level feature map. Therefore, we can use 3×3 convolutional layers to mine the spatial dependency between sensors with different distances. Another type of convolution with kernel size 1×1 (see Fig. 3(b)) maps an input pixel with all its channels and does not account for pixels around it. This convolution can be used to model temporal features. It should be noted that we do not use subsampling like pooling layer but only convolutions to ensure that the input and output images of the convolutional layer have the same size.

As shown in He and Sun (2015), the total time complexity of all convolutional layers is

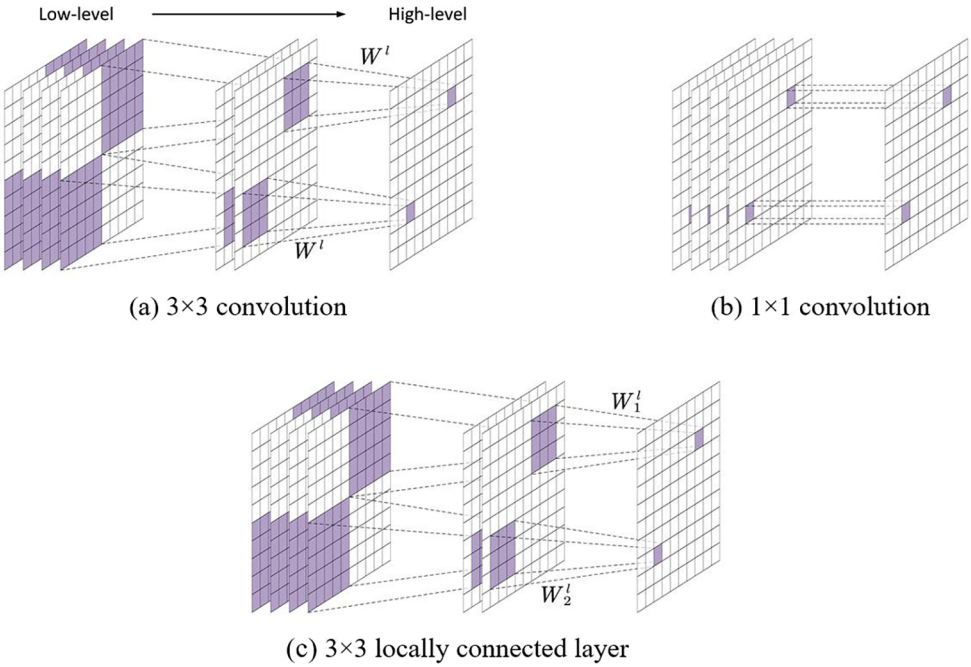

Fig. 3. Convolutional and locally connected layers. The convolutional layer shares weights when applied at each different patch of the input, but the locally connected layer does not share the weight.

$$O\left(\sum_{l=1}^{L} S_l^2 \cdot M_l^2 \cdot C_{l-1} \cdot C_l\right). \tag{9}$$

Here, l is the index of the convolutional layer, L is the number of convolutional layers, S_l is the spatial size (length) of the convolutional kernel. C_{l-1} and C_l are the numbers of kernels in the $(l-1)^{th}$ and l^{th} layers, respectively. M_l is the spatial size of the output feature map, and it is always equal to $\lceil \sqrt{n} \rceil$ in this study because we do not use subsampling and all feature maps have the same size as the input image. Therefore, Eq. (9) can be rewritten as

$$O\left(\sum_{l=1}^{L} S_l^2 \cdot n \cdot C_{l-1} \cdot C_l\right). \tag{10}$$

3.4. DeepTrend 2.0 model

We focus on designing a light-weighted model with fast training and good generation. Therefore, the proposed DeepTrend 2.0 adopts CNN to capture the spatial-temporal dependency of traffic flow and simple multi-task learning (Huang et al., 2014; Jin and Sun, 2008) strategy for multi-scale prediction.

The architecture of DeepTrend 2.0 is illustrated in Fig. 4. After obtaining the rearranged order of sensors, we turn the traffic flow in all the sensors in past N time intervals into an N-channel image $\mathbf{X} = [X_{t-(N-1)}, X_{t-(N-2)}, \cdots, X_t]$ using the method introduced in Section 3.2. Then, the image is fed into a hierarchical network with a locally connected layer and a stack of convolutional layers (i.e., Conv1 that is shown in Fig. 4) that can capture spatial features of the input image. It should be noted that the kernel of the locally connected layer and Conv1 should be of size larger than 1×1 to be capable of locally capturing spatial dependency, and the input image should be first padded with zero outside its border to guarantee that the output of the convolutional layer has the same size as the input. The obtained feature map from the top Conv1 is fed into four separate 1×1 convolutional layers (i.e., Conv2 that is shown in Fig. 4), which simultaneously yield one-channel images that represent the predicted flow in 5, 15, 30, and 45 min. The cost function of the model is the sum of square errors between predicted flow and true flow of four tasks.

Multi-task learning is implemented by sharing the hidden layers between all the tasks while keeping several task-specific output layers. It can obtain improved generalization through averaging the data-dependent noise of traffic flow and focus its attention on those features that are important for future traffic flow.

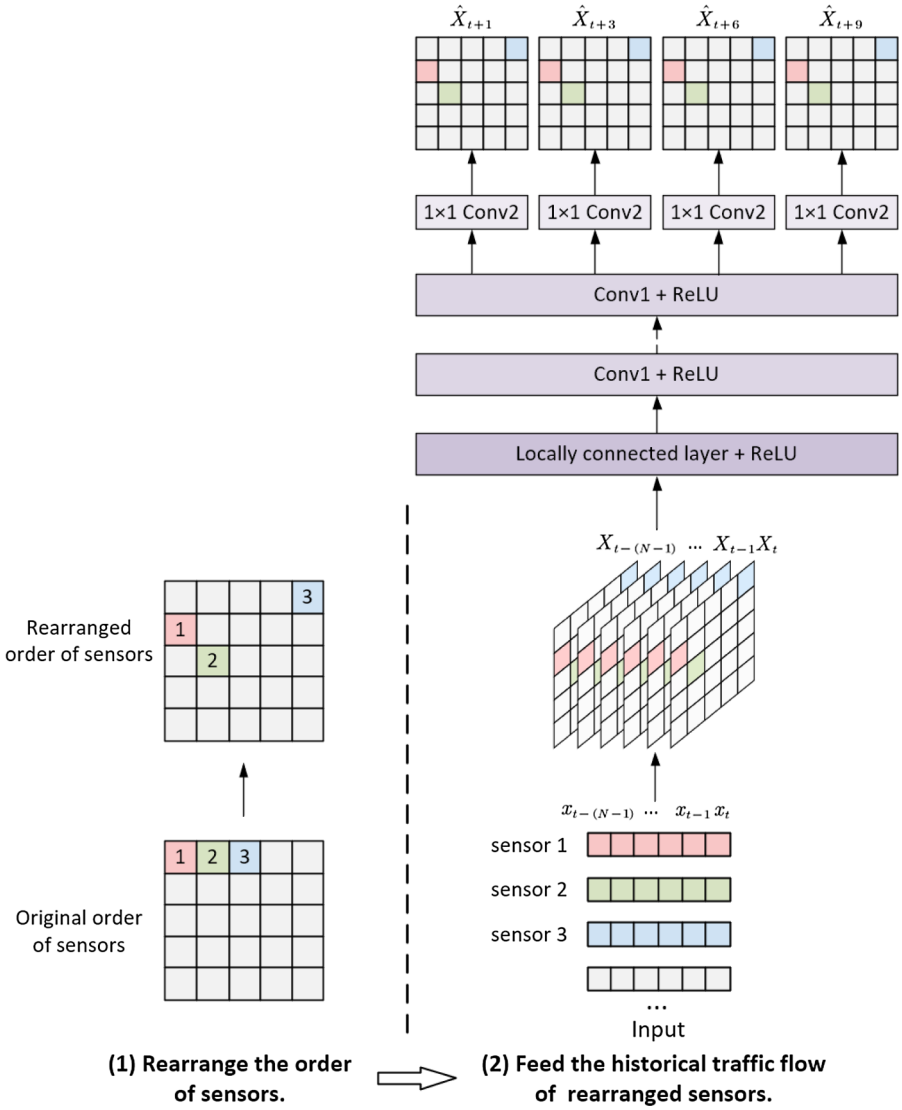

Fig. 4. Structure of DeepTrend 2.0. Conv1 and Conv2 represent convolutional layers.

4. Numerical testing results

4.1. Data sources and data-pre-processing

The experimental traffic flow data are obtained from the Caltrans Performance Measurements Systems (PeMS) traffic dataset (PeMS, 2016), which is one of the most widely used datasets for traffic prediction. Sensors in the PeMS dataset are loop detectors that span the freeway across all major metropolitan areas of the state of California, USA. A sensor with unique ID aggregates multiple lane loop detectors, and the flow value of a sensor station in each time step is equal to the total number of vehicles over a 5 min period across all the lanes. In this study, we obtain the traffic data collected from 3941 sensors in district 4 of the freeway systems across California in 2016 and remove the data of the holidays. Then, we filter out the sensors whose missing data are more than 5% and focus only on the remaining 380 sensors. The missing data of 380 sensors are imputed by the simple average trend method.

The spatial distribution of these sensors is shown in Fig. 5. Owing to limited computational resources, in the experiments, the proposed model is applied to the data during the first 16 weeks of 2016. The data of the first 12 weeks are used as the training set and that of the remaining 4 weeks as the test set.

During the training process, the original five min interval flow data and the residual data that are used in original methods and detrending methods are normalized to zero mean and unit variance before they are fed into the model. In the test process, the predicted data are scaled back to the original representation for performance comparison.

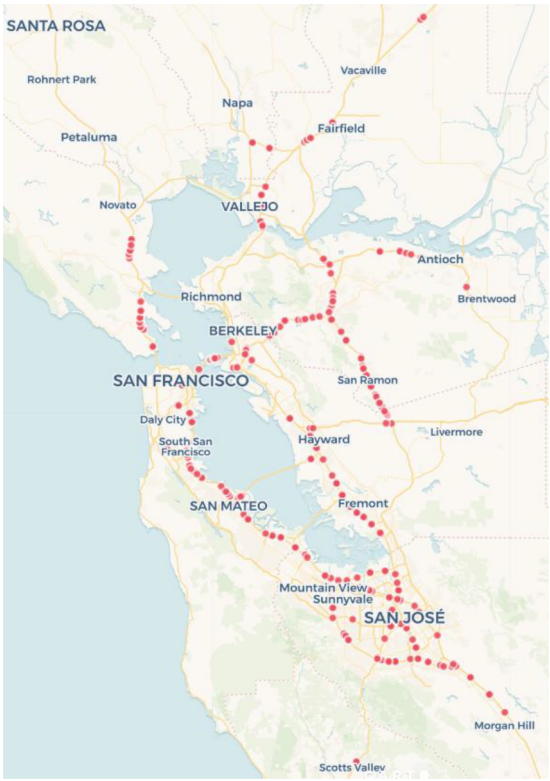

Fig. 5. Sensor distribution in district 4 of PeMS. Each red dot represents a sensor station. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.2. Performance indexes

In this study, the root mean square error (RMSE), mean absolute error (MAE) and mean absolute percentage error (MAPE) are utilized to measure the performance of traffic flow forecasting. These are defined as follows:

$$RMSE = \sqrt{\frac{1}{N_s}\sum_{n_s=1}^{N_s}(y_{n_s} - \hat{y}_{n_s})^2}, \tag{11}$$

$$MAE = \frac{1}{N_s}\sum_{n_s=1}^{N_s}|y_{n_s} - \hat{y}_{n_s}|, \tag{12}$$

$$MAPE = \frac{1}{N_s}\sum_{n_s=1}^{N_s}\frac{|y_{n_s} - \hat{y}_{n_s}|}{y_{n_s}}, \tag{13}$$

where y_{n_s} and $\hat{y}_{n_s}$ are the actual and the predicted traffic flows, respectively, and N_s is the number of test samples. For comparison of the total prediction performance between different models, the flow samples of entire sensors in the test set are used, whereas the samples of a sensor are used to measure the prediction errors of the models for the sensor.

4.3. Baselines

4.3.1. Graph convolutional neural network (GCNN)

Similar to CNN, GCNN (Defferrard et al., 2016; Kipf and Welling, 2017) comprises a stack of graph convolutional layers, where each layer comprises localized graph filters. The graph filters, which are restricted to operate on the neighboring nodes, are utilized to extract spatial features on graphs. To implement the graph convolution operation by graph filters, we treat the sensor network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where $\mathcal{V}$ is a set of $|\mathcal{V}| = n$ nodes, i.e., sensor stations, $\mathcal{E}$ is a set of edges, and $\mathbf{W} \in \mathbb{R}^{n\times n}$ is a weighted adjacency matrix that is defined by the correlation matrix of sensors.

Given a vector signal $x_g \in \mathbb{R}^n$ defined on the graph, we can get the spectral graph convolution on the signal by a filter, which is formed by polynomials in the Fourier domain. The graph Fourier transform of x_g is defined as the expansion of x_g in terms of the graph Laplacian L obtained from $\mathbf{W}$.

In GCNN based traffic prediction, historical traffic flow is represented as the graph signal $\mathbf{X_g} = [X_g^1, X_g^2, \cdots, X_g^N] \in \mathbb{R}^{n\times N}$ with an N-dimensional feature vector for each node. For a graph convolution with filter tensor Θ of size $[N, D_{out}, K_s]$, we can obtain the output feature map $\mathbf{H} = [H_1, H_2, \cdots, H_{D_{out}}]$, in which the j_g^{th} feature map is given by

$$H_{j_g} = f\left(\sum_{i_g=1}^{N}\sum_{k_g=0}^{K_s-1}\Theta_{i_g j_g k_g} L^{k_g} X_g^{i_g}\right) \in \mathbb{R}^n, \tag{14}$$

where f is the activation function. Additionally, the details about the graph convolution operator are proposed in Defferrard et al. (2016) and Kipf and Welling (2017).

In our experiment, we use the GCNN with a structure similar to DeepTrend 2.0 as the baseline. In other words, GCNN comprises a stack of graph Conv1 layers with four graph Conv2 layers on the top to predict the traffic flow of four time scales, and the parameters of each layer correspond to those of DeepTrend 2.0.

As GCNN is sensitive to the weighted adjacency matrix of the graph, and the number of non-zero elements in the matrix influence the complexity, we decrease the complexity of GCNN by introducing a threshold p in the correlation matrix that determines the number of edges in a graph. The correlation matrix for GCNN is based on the Spearman rank correlation coefficient and defined as:

$$Corr_g(s_k, s_l) = \begin{cases} |r(s_k, s_l)|, & \text{if } |r(s_k, s_l)| \geqslant p \\ 0, & \text{if } |r(s_k, s_l)| < p \end{cases}. \tag{15}$$

As shown in Kipf and Welling (2017), the total time complexity of all graph convolutional layers is

$$O\left(\sum_{l_g=1}^{L_g} |\mathcal{E}|\cdot C_{l_g-1}\cdot C_{l_g}\right), \tag{16}$$

where l_g is the index of the graph convolutional layer, L_g is the number of graph convolutional layers, $|\mathcal{E}|$ is the number of edges of the graph, and C_{l_g-1} and C_{l_g} are the number of graph filters in the $(l_g - 1)^{th}$ and l_g^{th} layers, respectively. $|\mathcal{E}|$ has a linear relationship with the number of non-zero elements in the correlation matrix denoted as $m_{nonzero}$; therefore, Eq. (16) can be rewritten as

$$O\left(\sum_{l_g=1}^{L_g} m_{nonzero}\cdot C_{l_g-1}\cdot C_{l_g}\right). \tag{17}$$

4.3.2. Long short-term memory network (LSTM)

LSTM (Hochreiter and Schmidhuber, 1997) is a special kind of recurrent neural network (RNN) (Goodfellow et al., 2016). RNN is a generation of the feedforward neural network and is designed to deal with sequence data. Given an input sequence, the hidden state in RNN is calculated at each time step by a function of the input at the current time and the hidden state at the previous time, generating a hidden sequence. The output can be derived from a function of the last hidden state or the hidden sequence. However, the basic RNN is exposed to the risk of vanishing or exploding gradients during training due to the plain structure. LSTM is proposed to tackle these problems. It uses a particular structure named LSTM units, which contain an input gate, an output gate, a forget gate in hidden layer to avoid the vanishing or exploding gradients while capturing long-term temporal dependencies. To implement LSTM, the traffic flow in all sensors at each time step are formulated as a vector, and vectors in previous time intervals are fed into LSTM. Finally, four linear fully connected layers which share the hidden layer in LSTM will output four vectors representing the future traffic flow of all sensors in four time scales.

4.4. Parameter settings

In the formulation of many-to-many traffic flow prediction problem, the correlation matrix is obtained by the original traffic series for original data based prediction. For detrending based prediction, we implement two kinds of correlation matrices calculated by original data and residual data to rearrange the order of sensors. For the rearrangement algorithm, parameter α is set to 3 to determine the neighbors of the pixel.

For DeepTrend 2.0, we conduct a grid search to determine the optimal parameters on the validation set. In this study, we select

Table 1
Parameter selection in DeepTrend 2.0. (Ori: original data based model; Det-O: detrending based model and correlation matrix calculated by original data; Det-R: detrending based model and correlation matrix calculated by residual data.)

Parameters	Search space	Optimal parameters (Ori)	Optimal parameters (Det-O)	Optimal parameters (Det-R)
Filter size of Conv1 and fully connected layers	{3 × 3, 4 × 4, 5 × 5}	5 × 5	3 × 3	3 × 3
Number of kernels in each hidden layer	{32, 64, 128}	128	128	128
Number of hidden layers	{1, 2, 3, 4}	2	3	4
Time lag	{9, 12, 15, 18}	12	9	9

Table 2
Performance comparison of original data (Ori) and detrending based (Det-O, Det-R) DeepTrend 2.0 in predicting traffic flow of four time scales. Test results are selected under the best parameters for each method. The best results are highlighted in bold. Percentage changes of detrending based methods over original data based method are shown in parenthesis.

	Time scale	5 min	15 min	30 min	45 min
Ori	RMSE	**27.2**	30.0	32.6	34.5
	MAE	18.2	20.0	21.8	23.2
	MAPE	0.124	0.128	0.147	0.152
Det-O	RMSE	**27.2 (−0.0%)**	29.7 (−1.0%)	31.5 (−3.4%)	32.6 (−5.5%)
	MAE	**17.9 (−1.6%)**	**19.3 (−3.5%)**	20.3 (−6.9%)	**21.0 (−9.5%)**
	MAPE	**0.105 (−15.3%)**	**0.113 (−11.7%)**	**0.117 (−20.4%)**	0.121 (−20.4%)
Det-R	RMSE	**27.2 (−0.0%)**	**29.6 (−1.3%)**	**31.3 (−4.0%)**	**32.5 (−5.8%)**
	MAE	**17.9 (−1.6%)**	**19.3 (−3.5%)**	**20.2 (−7.3%)**	**21.0 (−9.5%)**
	MAPE	0.108 (−12.9%)	**0.113 (−11.7%)**	**0.117 (−20.4%)**	**0.119 (−21.7%)**

90% of the training data to train the model, and the remaining 10% is chosen as the validation set. Early stopping, which stops the training algorithm if there is no increase in validation score in several iterations, is used during training. Table 1 summarizes the four parameters that we need to locate by the grid search in the search space. The first three parameters determine the architecture of DeepTrend 2.0, and the last parameter determines the historical time window.

Optimal parameters for the original data based and detrending based models are listed in the last three columns in Table 1, where Ori represents the original data based prediction, Det-O represents the detrending based prediction with the correlation matrix calculated by original data, and Det-R represents the detrending based prediction with the correlation matrix calculated by residual data. Additionally, some other parameters of DeepTrend 2.0 need to be set. The activation function in the hidden layers is ReLU (Nair and Hinton, 2010), and there is a linear function in the output layer. The Adam algorithm (Kingma and Ba, 2015) is used to train the model, and the learning rate is 0.001. The batch size is set to 128.

GCNN and LSTM models have similar parameters as DeepTrend 2.0. The original data based GCNN and LSTM both comprise 2 hidden layers with 128 filters each, and the former uses Chebyshev polynomials approximation, where the order of the polynomial is 3. The detrending based GCNN and LSTM both comprise 4 hidden layers, and the order of the polynomial in GCNN is 1; other parameters are the same as that of their original data based models. Additionally, the threshold p in Eq. (15) is set to 0.9 for original data based GCNN, and 0.2 for detrending based GCNN.

We use the Keras (Chollet, 2015) library with Tensorflow (Abadi et al., 2016) as the backend to build DeepTrend 2.0 and the GCNN. All experiments were conducted on a Linux server (CPU: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz, GPU: NVIDIA TITAN V).

4.5. Performance of detrending based DeepTrend 2.0 and rearrangement algorithm

Table 2 shows the results of the original data based (Ori) and detrending based (Det-O, Det-R; Det-O denotes the correlation matrix calculated by original data, and Det-R denotes the correlation matrix is calculated by residual data) DeepTrend 2.0 in predicting the traffic flow of four time scales under optimal parameters. It is seen that the detrending based methods that use residual data to make predictions, significantly boost the prediction performance in all three evaluation metrics compared with the original data based method. Especially, detrending has a considerable advantage in long-term (30 and 45 min) prediction, because it handles the long-term trend and only needs to account for the residual data. We can also find that the performance of Det-R is slightly better than Det-O, which shows that it is more appropriate to use residual data to calculate the correlation matrix for detrending based DeepTrend 2.0.

To illustrate the benefits of detrending further, we compare the performance of original data and detrending based DeepTrend 2.0 for each sensor (Fig. 6). Comparison results show that detrending boosts the prediction performance for most sensors, and the enhancement is significant on large time scales. This indicates that detrending is useful in most cases and is beneficial for deep learning models.

Fig. 7 shows the performance of point prediction under original data (Ori) and detrending (Det-R) based methods on a typical day. The real traffic flow data is selected in sensor with ID 400119 on April 25, 2016. It can be seen from Fig. 7 that the predicted values of Det-R could track the real values of the traffic flow satisfactorily, whereas those of Ori have large deviations from the true values for some time periods like 16:00–18:00. This indicates the workability of the detrending based models.

We further test the prediction results for all combinations of the parameters in the search space in Table 1 for original data (Ori) based and detrending (Det-R) based methods. Both the methods were tested 144 times under different combinations of the parameters. Fig. 8 illustrates the test results, wherein the height of bars represents the average of the results, and the error bar represents the standard deviation. It can be observed that in a statistical sense, the detrending based method under different combinations of model parameters achieves better performance than the original data based method. Besides, the standard deviation of the test results for the detrending based methods is negligible, whereas that for the original data based methods is high. This proves that the prediction for the residual data is insensitive to model architecture. In conclusion, the detrending based method is effective and robust for many-to-many traffic prediction.

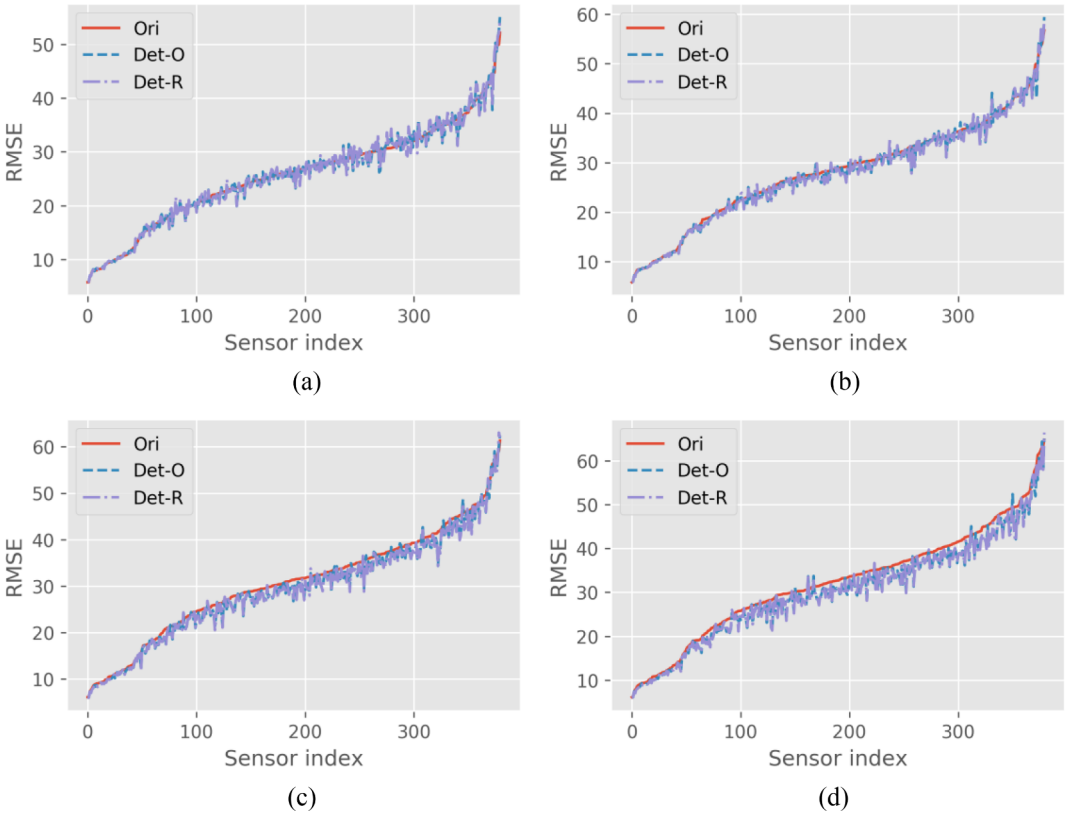

Fig. 6. Performance comparison of original data (Ori) and detrending based (Det-O, Det-R) DeepTrend 2.0 for each sensor (total 380). Sensors have been sorted in ascending order of their original data based prediction errors. Each subfigure represents comparison results on a time scale. (a) 5 min; (b) 15 min; (c) 30 min; (d) 45 min.

To illustrate the significance of the proposed rearrangement algorithm, which was introduced in Section 3.2, we compare the prediction performance for sensor orders with different total scores under original data based and detrending based methods. We test 5 sensor arrangements with different scores in which each arrangement is tested for 20 times. The comparison boxplot is shown in Fig. 9. As depicted in Fig. 9(a), larger total scores bring benefits for short-term (5 min and 15 min) prediction under original data based methods. This indicates the advantage of rearrangement for image-based traffic prediction. We also find that the performance of long-term (30 min and 45 min) prediction does not significantly relate to total scores. This may be caused by substantial uncertainty in long-term prediction, which is hard to handle by original data even after rearrangement. For detrending based models shown in Fig. 9(b), prediction accuracy increases with the growth of total scores under four time scales. This indicates that the correlation of sensors is the key to residual series prediction. To sum up, the rearranged-ordered image, wherein pixels of highly correlated sensors are close, is beneficial to CNN models for improved prediction especially under detrending methods.

4.6. Comparison with GCNN based prediction model

Fig. 10 illustrates the performance comparison of DeepTrend 2.0, LSTM and GCNN based on original data (Ori) and detrending (Det). Here, Det for DeepTrend 2.0 and GCNN represents detrending based prediction with the correlation matrix calculated by residual data, i.e., Det-R in our previous definition. It can be seen that detrending yields higher prediction accuracy than original data based methods for DeepTrend 2.0 and the GCNN models, which shows that detrending is advantageous for these state-of-the-art models. We also find that detrending is not very useful for LSTM, which shows that LSTM may not adapt to high dimensional residual data. Furthermore, if we consider detrending, the prediction accuracy for both DeepTrend 2.0 and GCNN is very similar and is significantly superior to that of LSTM. This means that the proposed CNN model can adequately handle the spatial-temporal dependency of the sensor network like the GCNN model under detrending, while LSTM is insufficient to deal with it.

We further compare the training and inference times consumed by both DeepTrend 2.0 and GCNN. As seen in Table 3, the proposed DeepTrend 2.0 costs less than a tenth of the training time of the GCNN under detrending based methods, and an eighth of training time of GCNN under original data based methods. Moreover, detrending based DeepTrend 2.0 costs only 6.6 s per training iteration, whereas GCNN costs 17.4 s. This suggests that our model based on CNN can achieve faster convergence during training than GCNN. Additionally, the inference time of DeepTrend 2.0 is significantly less than that of GCNN, indicating that the proposed model is more suitable for real-time applications. To sum up, the detrending based DeepTrend 2.0 makes trade-off between accuracy and

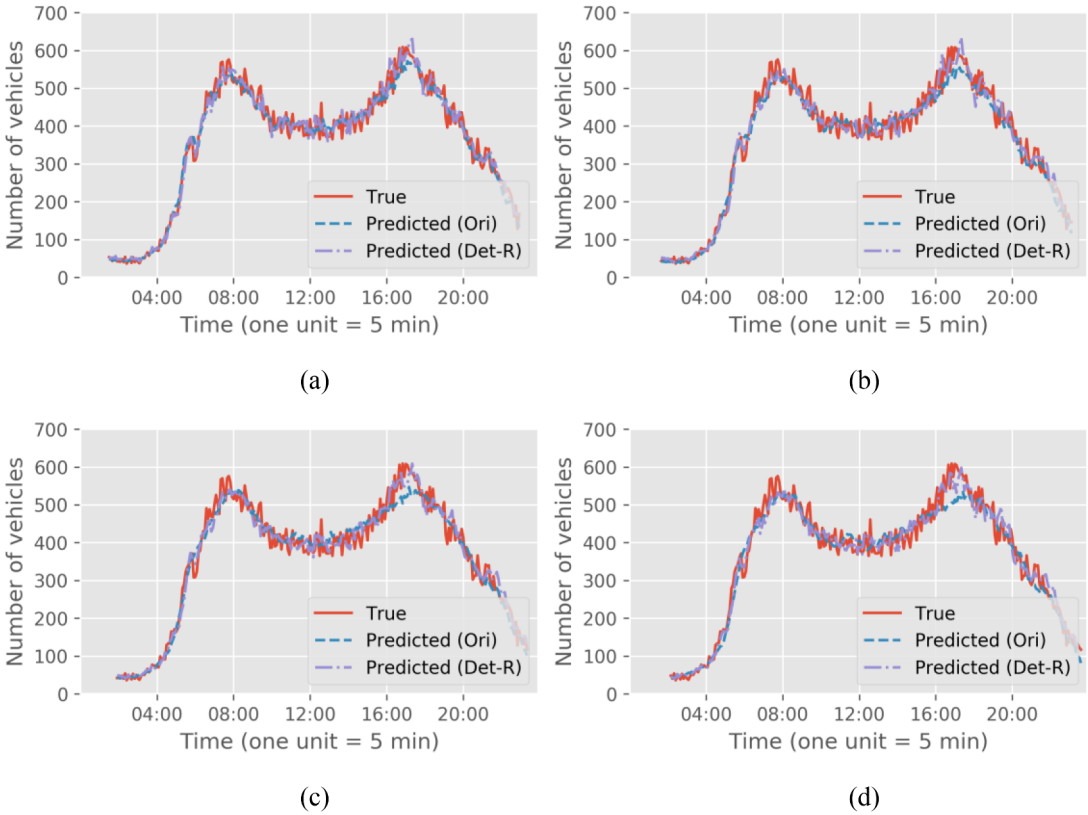

Fig. 7. True and predicted flow of sensor with ID 400119 on April 25, 2016. Each subfigure represents the comparison results on a time scale. (a) 5 min; (b) 15 min; (c) 30 min; (d) 45 min.

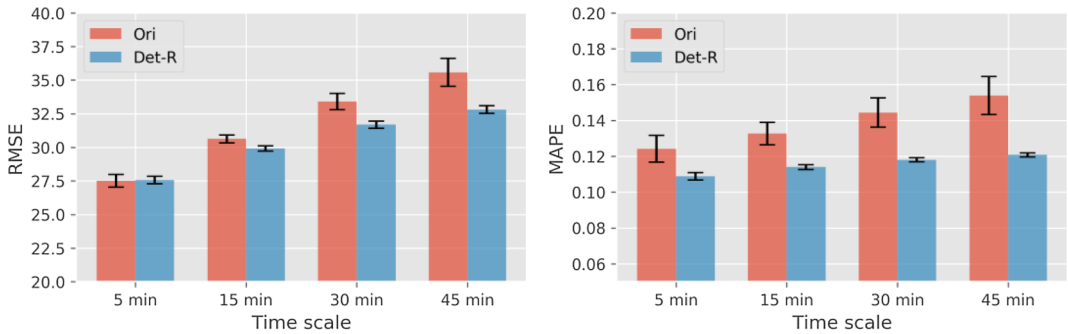

Fig. 8. Performance comparison of original data (Ori) and detrending (Det-R) based DeepTrend 2.0. Tests for both the methods were conducted for 144 times under different combinations of parameters.

time and is easy to transfer and deploy.

5. Conclusions

In this paper, we introduce detrending into many-to-many prediction and propose a light-many-to-many prediction model: DeepTrend 2.0. The following conclusions can be drawn from the tests on empirical data:

First, detrending may prove to be a useful data-preprocessing method for traffic prediction irrespective of the type of deep learning models used because it handles the global pattern in traffic flow and magnifies the local trend. It will prove to be advantageous for models regarding their prediction performance, help with robustness, and speed up convergence. Therefore, we recommend testing the potential benefits of detrending for traffic prediction because it is worth testing without being costly in the measure of time or money.

Second, it would be beneficial to maintain a good balance between model complexity and accuracy, especially when complex deep learning based traffic prediction models are considered. Practical implementation efficiency needs to be addressed carefully,

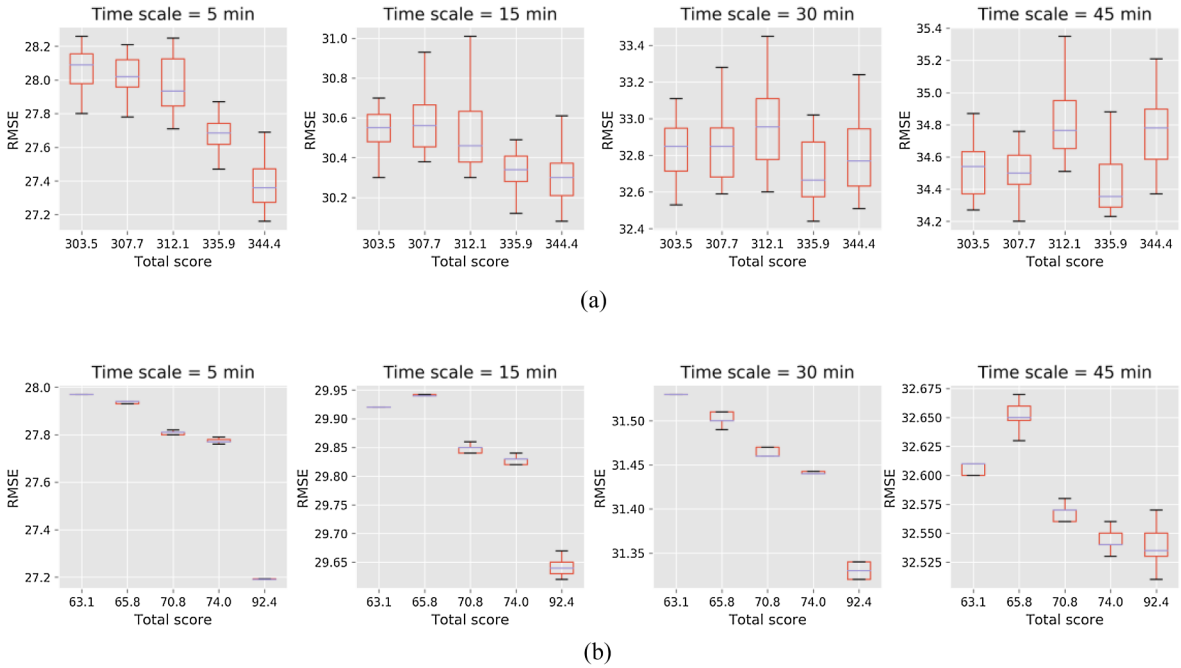

Fig. 9. Performance comparison of sensor orders with different scores for original data based and detrending based methods. (a) Original data (O). (b) Detrending (Det-R).

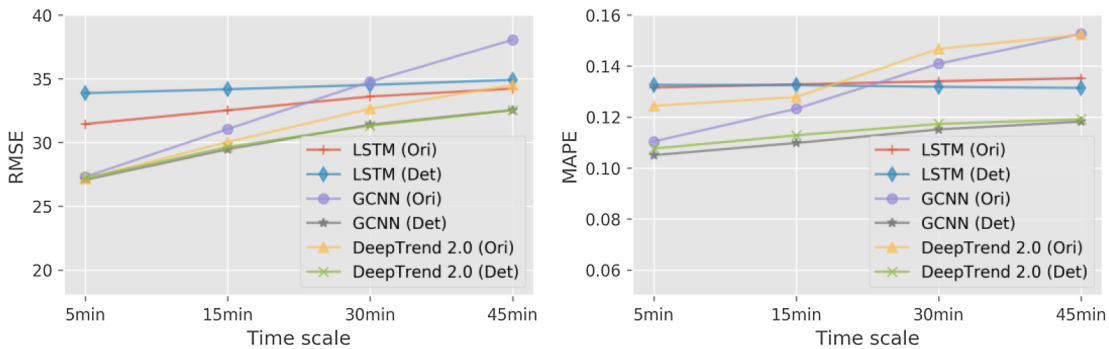

Fig. 10. Performance comparison of DeepTrend 2.0, LSTM and GCNN based on original data (Ori) and detrending (Det-R).

Table 3
Training and inference time comparison of GCNN and DeepTrend 2.0.

Model	Number of iterations	Total training time (s)	Average training time per iteration (s)	Inference time (s)
GCNN (Ori)	25	575	23.0	3.3
GCNN (Det)	21	366	17.4	2.6
DeepTrend 2.0 (Ori)	13	**89**	**6.8**	**0.8**
DeepTrend 2.0 (Det)	5	**33**	**6.6**	**0.8**

The training and inference time of DeepTrend 2.0 are highlighted in bold.

especially recently, when the improvement of prediction accuracy has become increasingly difficult.

For future research, we plan to apply such a light-weighted model for some other related studies such as travel time prediction and traffic demand prediction, and further consider both the geographical relationship and similarity between sensors for better spatial-temporal modeling. Moreover, we would like to discuss an improved method to integrate detrending into deep learning models to achieve better trade-off between model complexity and accuracy.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under Grants 61533019 and U1811463, the Beijing Municipal Science and Technology Commission Program under Grant D171100000317002, and the Beijing Municipal Commission of Transport Program under Grant ZC179074Z.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., et al., 2016. TensorFlow: large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467.
Ahmed, M.S., Cook, A.R., 1979. Analysis of freeway traffic time-series data by using box-jenkins techniques. Transp. Res. Rec. 1–9.
Boto-Giralda, D., Díaz-Pernas, F.J., González-Ortega, D., Díez-Higuera, J.F., Antón-Rodríguez, M., Martínez-Zarzuela, M., Torre-Díez, I., 2010. Wavelet-based de-noising for traffic volume time series forecasting with self-organizing neural networks. Comput.-Aided Civ. Infrastruct. Eng. 25, 530–545.
Cao, X., Zhong, Y., Zhou, Y., Wang, J., Zhu, C., Zhang, W., 2017. Interactive temporal recurrent convolution network for traffic prediction in data centers. IEEE Access 1–15.
Castro-Neto, M., Jeong, Y.-S., Jeong, M.-K., Han, L.D., 2009. Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions. Expert Syst. Appl. 36, 6164–6173.
Cetin, M., Comert, G., 2006. Short-term traffic flow prediction with regime switching models. Transport. Res. Rec.: J. Transport. Res. Board 1965, 23–31.
Chen, C., Wang, Y., Li, L., Hu, J., Zhang, Z., 2012. The retrieval of intra-day trend and its influence on traffic prediction. Transport. Res. Part C: Emerg. Technol. 22, 103–118.
Cheng, X., Zhang, R., Zhou, J., Xu, W., 2017. DeepTransport: Learning spatial-temporal dependency for traffic condition forecasting. arXiv:1709.09585.
Chollet, F., 2015. Keras. < https://github.com/fchollet/keras > .
Cui, Z., Henrickson, K., Ke, R., Wang, Y., 2018. High-order graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. arXiv:1802.07007.
Dai, X., Fu, R., Lin, Y., Li, L., Wang, F.-Y., 2017. DeepTrend: A deep hierarchical neural network for traffic flow prediction. arXiv:1707.03213.
Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of 30th Conference on Neural Information Processing Systems, pp. 3844–3852.
Dixon, M.F., Polson, N.G., Sokolov, V.O., 2018. Deep learning for spatio-temporal modeling: dynamic traffic flows and high frequency trading. Appl. Stochastic Models Bus. Indus.
Fu, R., Zhang, Z., Li, L., 2016. Using lstm and gru neural network methods for traffic flow prediction. In: Proceedings of the 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), pp. 324–328.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.
He, K., Sun, J., 2015. Convolutional neural networks at constrained time cost. In: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5353–5360.
He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 770–778.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780.
Huang, W., Song, G., Hong, H., Xie, K., 2014. Deep architecture for traffic flow prediction: deep belief networks with multitask learning. IEEE Trans. Intell. Transp. Syst. 15, 2191–2201.
Jin, F., Sun, S., 2008. Neural network multitask learning for traffic flow forecasting. In: Proceedings of 2008 IEEE International Joint Conference on Neural Networks, pp. 1897–1901.
Jin, X., Zhang, Y., Yao, D., 2007. Simultaneously prediction of network traffic flow based on pca-svr. In: Advances in Neural Networks – ISNN 2007. Lecture Notes in Computer Science, vol. 4492, Springer, Berlin, Heidelberg.
Kamarianakis, Y., Shen, W., Wynter, L., 2012. Real-time road traffic forecasting using regime-switching space-time models and adaptive lasso. Appl. Stochastic Models Bus. Ind. 28, 297–315.
Kim, Y., Wang, P., Zhu, Y., Mihaylova, L., 2018. A capsule network for traffic speed prediction in complex road networks. arXiv:1807.10603.
Kingma, D.P., Ba, J., 2015. Adam: A method for stochastic optimization. In: Proceedings of International Conference on Learning Representations.
Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: Proceedings of International Conference on Learning Representations.
Koesdwiady, A., Soua, R., Karray, F., 2016. Improving traffic flow prediction with weather information in connected cars: a deep learning approach. IEEE Trans. Veh. Technol. 65, 9508–9517.
Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324.
Leshem, G., Ritov, Y., 2007. Traffic flow prediction using adaboost algorithm with random forests as a weak learner. Int. J. Intell. Technol. 2, 111–116.
Levin, M., Tsao, Y.-D., 1980. On forecasting freeway occupancies and volumes. Transp. Res. Rec. 47–49.
Li, L., Su, X., Wang, Y., Lin, Y., Li, Z., Li, Y., 2015a. Robust causal dependence mining in big data network and its application to traffic flow predictions. Transp. Res. Part C: Emerg. Technol. 58, 292–307.
Li, L., Su, X., Zhang, Y., Lin, Y., Li, Z., 2015b. Trend modeling for traffic time series analysis: an integrated study. IEEE Trans. Intell. Transp. Syst. 16, 3430–3439.
Li, Y., Yu, R., Shahabi, C., Liu, Y., 2017a. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. arXiv:1707.01926.
Li, Z., Jiang, S., Li, L., Li, Y., 2017b. Building sparse models for traffic flow prediction: an empirical comparison between statistical heuristics and geometric heuristics for bayesian network approaches. Transp. B: Transp. Dyn. 1–17.
Li, Z., Li, Y., Li, L., 2014. A comparison of detrending models and multi-regime models for traffic flow prediction. IEEE Intell. Transp. Syst. Mag. 6, 34–44.
Lin, L., He, Z., Peeta, S., Wen, X., 2017. Predicting station-level hourly demands in a large-scale bike-sharing network: A graph convolutional neural network approach. arXiv:1712.04997.
Lin, Y., Dai, X., Li, L., Wang, F.-Y., 2018. Pattern sensitive prediction of traffic flow based on generative adversarial framework. IEEE Trans. Intell. Transp. Syst. https://doi.org/10.1109/TITS.2018.2857224.
Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.-Y., 2014. Traffic flow prediction with big data: a deep learning approach. IEEE Trans. Intell. Transp. Syst. 16, 865–873.
Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., Wang, Y., 2017. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. Sensors 17, 818.
Ma, X., Tao, Z., Wang, Y., Yu, H., Wang, Y., 2015a. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. Transp. Res. Part C: Emerg. Technol. 54, 187–197.
Ma, X., Yu, H., Wang, Y., Wang, Y., 2015b. Large-scale transportation network congestion evolution prediction using deep learning theory. Plos One 10, 1–17.
Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning, pp. 807–814.
Okutani, I., Stephanedes, Y.J., 1984. Dynamic prediction of traffic volume through kalman filtering theory. Transp. Res. Part B: Methodol. 18, 1–11.
Pang, L., Lan, Y., Xu, J., Guo, J., Cheng, X., 2017. Locally smoothed neural networks. arXiv:1711.08132 [cs, stat].
PeMS, 2016. Caltrans Performance Measurement System. < http://pems.dot.ca.gov/ > .
Polson, N.G., Sokolov, V.O., 2017. Deep learning for short-term traffic flow prediction. Transp. Res. Part C: Emerg. Technol. 79, 1–17.
Qu, L., Li, L., Zhang, Y., Hu, J., 2009. PPCA-based missing data imputation for traffic flow volume: a systematical approach. IEEE Trans. Intell. Transp. Syst. 10,

512–522.
Tan, H., Xuan, X., Wu, Y., Zhong, Z., Ran, B., 2016. A comparison of traffic flow prediction methods based on dbn. In: Proceedings of the 16th COTA International Conference of Transportation Professionals, pp. 273–283.
Vlahogianni, E.I., Karlaftis, M.G., Golias, J.C., 2014. Short-term traffic forecasting: where we are and where we're going. Transp. Res. Part C: Emerg. Technol. 43, 3–19.
Williams, B.M., Hoel, L.A., 2003. Modeling and forecasting vehicular traffic flow as a seasonal arima process: theoretical basis and empirical results. J. Transp. Eng. 129, 664–672.
Wu, Y., Tan, H., Qin, L., Ran, B., Jiang, Z., 2018. A hybrid deep learning based traffic flow prediction method and its understanding. Transp. Res. Part C: Emerg. Technol. 90, 166–180.
Yang, G., Wang, Y., Yu, H., Ren, Y., Xie, J., 2018. Short-term traffic state prediction based on the spatiotemporal features of critical road sections. Sensors 18, 2287.
Yu, B., Yin, H., Zhu, Z., 2017a. Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting. arXiv:1709.04875.
Yu, H., Wu, Z., Wang, S., Wang, Y., Ma, X., 2017b. Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. Sensors (Basel) 17, 1501.
Yu, R., Li, Y., Shahabi, C., Demiryurek, U., Liu, Y., 2017c. Deep learning: a generic approach for extreme condition traffic forecasting. In: Proceedings of the 2017 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, pp. 777–785.
Zhang, J., Zheng, Y., Qi, D., 2017a. Deep spatio-temporal residual networks for citywide crowd flows prediction. Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17).
Zhang, J., Zheng, Y., Qi, D., Li, R., Yi, X., Li, T., 2017b. Predicting citywide crowd flows using deep spatio-temporal residual networks. arXiv:1701.02543.
Zhao, Z., Chen, W., Wu, X., Chen, P.C.Y., Liu, J., 2017. LSTM network: a deep learning approach for short-term traffic forecast. IET Intel. Transport Syst. 11, 68–75.
Zwillinger, D., Kokoska, S., 1999. CRC Standard Probability and Statistics Tables and Formulae. CRC Press.