Predicting citywide crowd flows using deep spatio-temporal residual networks ☆

Junbo Zhang [a], Yu Zheng [b,c,d,e,*,1], Dekang Qi [c,2], Ruiyuan Li [d,2], Xiuwen Yi [c,2], Tianrui Li [c]

[a] Urban Computing Group, Microsoft Research, Beijing, China
[b] Urban Computing Business Unit, JD Finance, Beijing, China
[c] School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China
[d] School of Computer Science and Technology, Xidian University, China
[e] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China

ARTICLE INFO

ABSTRACT

Forecasting the flow of crowds is of great importance to traffic management and public safety, and very challenging as it is affected by many complex factors, including spatial dependencies (nearby and distant), temporal dependencies (closeness, period, trend), and external conditions (e.g. weather and events). We propose a deep-learning-based approach, called ST-ResNet, to *collectively* forecast two types of crowd flows (i.e. inflow and outflow) in each and every region of a city. We design an end-to-end structure of ST-ResNet based on unique properties of spatio-temporal data. More specifically, we employ the residual neural network framework to model the temporal closeness, period, and trend properties of crowd traffic. For each property, we design a branch of residual convolutional units, each of which models the spatial properties of crowd traffic. ST-ResNet learns to dynamically aggregate the output of the three residual neural networks based on data, assigning different weights to different branches and regions. The aggregation is further combined with external factors, such as weather and day of the week, to predict the final traffic of crowds in each and every region. We have developed a real-time system based on *Microsoft Azure Cloud*, called UrbanFlow, providing the crowd flow monitoring and forecasting in Guiyang City of China. In addition, we present an extensive experimental evaluation using two types of crowd flows in Beijing and New York City (NYC), where ST-ResNet outperforms nine well-known baselines.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Predicting crowd flows in a city is of great importance to traffic management, risk assessment, and public safety [2]. For instance, massive crowds of people streamed into a strip region at the 2015 New Year's Eve celebrations in Shanghai, result-

---

☆ This paper is an extended version of [1], which has been accepted for presentation at the 31st AAAI Conference on Artificial Intelligence (AAAI-17).
* Corresponding author at: Urban Computing Business Unit, JD Finance, Beijing, China.
 E-mail addresses: junbo.zhang@microsoft.com, msjunbozhang@outlook.com (J. Zhang), msyuzheng@outlook.com (Y. Zheng), dekangqi@outlook.com (D. Qi), v-ruiyli@microsoft.com (R. Li), v-xiuyi@microsoft.com (X. Yi), trli@swjtu.edu.cn (T. Li).
[1] Partial work was done when Yu Zheng was a senior researcher at Microsoft Research.
[2] The research was done when the third, fourth and fifth authors were interns at Microsoft Research.

(a) Inflow and outflow   (b) Measurement of flows
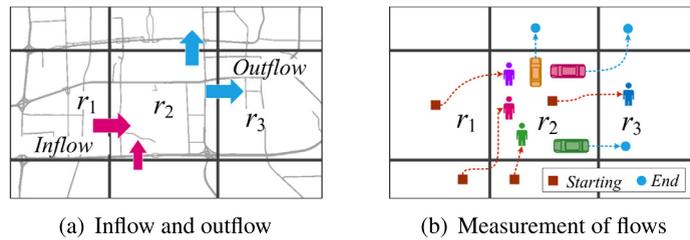
Fig. 1. Crowd flows in a region.

ing in a catastrophic stampede that killed 36 people. In mid-July of 2016, hundreds of "Pokemon Go" players ran through New York City's Central Park in hopes of catching a particularly rare digital monster, leading to a dangerous stampede there. If one can predict the crowd flow in a region, such tragedies can be mitigated or prevented by utilizing emergency mechanisms, such as conducting traffic control, sending out warnings, or evacuating people, in advance.

In this paper, we predict two types of crowd flows [3]: inflow and outflow, as shown in Fig. 1(a). Inflow is the total traffic of crowds entering a region from other places during a given time interval. Outflow denotes the total traffic of crowds leaving a region for other places during a given time interval. Both types of flows track the transition of crowds between regions. Knowing them is very beneficial for risk assessment and traffic management. Inflow/outflow can be measured by the number of pedestrians, the number of cars driven nearby roads, the number of people traveling on public transportation systems (e.g. metro, bus), or *all of them together* if data is available. Fig. 1(b) presents an illustration. We can use mobile phone signals to measure the number of pedestrians, showing that the inflow and outflow of r_2 are $(3, 1)$, respectively. Similarly, using the GPS trajectories of vehicles, two types of flows are $(0, 3)$, respectively. Therefore, the total inflow and outflow of r_2 are $(3, 4)$, respectively. Apparently, predicting crowd flows can be viewed as a kind of spatio-temporal prediction problem [2].

Deep learning [4] has been used successfully in many applications, and is considered to be one of the most cutting-edge artificial intelligence (AI) techniques. Exploring these techniques for spatio-temporal data is of great importance to a series of various spatio-temporal applications, including urban planning, transportation, the environment, energy, social, economy, public safety and security [2]. Although two main types of deep neural networks (DNNs) have considered a sort of spatial or temporal property: 1) convolutional neural networks (CNNs) for capturing spatial structures; 2) recurrent neural networks (RNNs) for learning temporal dependencies. It is still very challenging to apply these existing AI techniques for such spatio-temporal prediction problem because of the following three complex factors:

1. Spatial dependencies.

   Nearby   The inflow of Region r_2 (shown in Fig. 1(a)) is affected by outflows of *nearby* regions (like r_1). Likewise, the outflow of r_2 would affect inflows of other regions (e.g. r_3). The inflow of region r_2 would affect its own outflow as well.

   Distant   The flows can be affected by that of *distant* regions. For instance, people who lives far away from the office area always go to work by metro or highway, implying that the outflows of *distant* regions directly affect the inflow of the office area.

2. Temporal dependencies.

   Closeness   The flow of crowds in a region is affected by recent time intervals, both near and far. For instance, a traffic congestion occurring at 8:00am will affect that of 9:00am. And the crowd flows of today's 16th time interval[3] is more similar to that of yesterday's 16th time interval than that of today's 20th time interval.

   Period   Traffic conditions during morning rush hours may be similar on consecutive weekdays, repeating every 24 hours.

   Trend   Morning rush hours may gradually happen later as winter comes. When the temperature gradually drops and the sun rises later in the day, people get up later and later.

3. External influence. Some external factors, such as weather conditions and events may change the flow of crowds tremendously in different regions of a city. For example, a thunderstorm affects the traffic speed on roads and further changes the flows of regions.

---

[3]   Assume that half-hour is a time interval, 16th time interval means 7:30am–8:00am.

(a) Map segmentation   (b) Inflow matrix
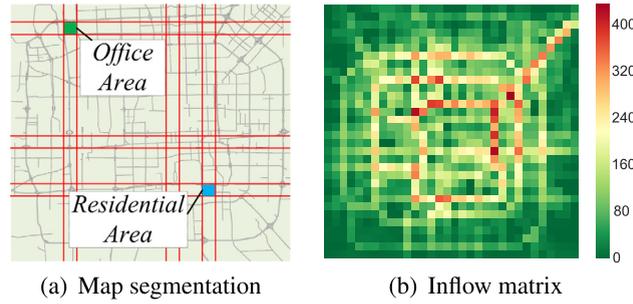
Fig. 2. Regions in Beijing: (a) Grid-based map segmentation; (b) inflows in every region of Beijing. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

To tackle above mentioned challenges, we here explore DNNs for spatio-temporal data, and propose a deep spatio-temporal residual network (ST-ResNet) to *collectively* predict inflow and outflow of crowds in every region. Our contributions are five-fold:

- ST-ResNet employs convolution-based residual networks to model both *nearby* and *distant* spatial dependencies between any two regions in a city, while ensuring the model's prediction accuracy is not comprised by the deep structure of the neural network.
- We summarize the temporal properties of crowd flows into three categories, consisting of temporal *closeness*, *period*, and *trend*. ST-ResNet uses three different residual networks to model these properties, respectively.
- ST-ResNet dynamically aggregates the output of the three aforementioned networks, assigning different weights to different branches and regions. The aggregation is further combined with external factors (e.g. weather).
- We evaluate our approach using Beijing taxicabs' trajectories and meteorological data, and NYC bike trajectory data. The results demonstrate the advantages of our approach compared with 9 baselines.
- We develop a real-time crowd flow monitoring and forecasting system using ST-ResNet. And our solution is based on the *Cloud* and GPU servers, providing powerful and flexible computational environments.

The rest of this paper is organized as follows. In Section 2, we formally describe the crowd flow prediction problem. Section 3 describes the proposed deep spatio-temporal residual networks for predicting crowd flows. Section 4 overviews the real-time crowd flow forecasting system based on ST-ResNet and Microsoft Azure Cloud. We present the evaluation in Section 5 and summarized the related work in Section 6.

The differences between this paper and our earlier work [1] are four aspects. First, we have deployed a cloud-based system that continuously forecasts the flow of taxicabs in each and every region of Guiyang City in China, showcasing the capability of ST-ResNet in handling real-world problems. The implementation of the cloud-based system is also introduced in Section 4 of this paper. Second, we extend ST-ResNet from a one-step ahead prediction to a multi-step ahead prediction, enabling the prediction of crowd flows over a farther future time (Section 3.4). Third, we conduct more comprehensive experiments on crowd flow prediction, showcasing the effectiveness and robustness of ST-ResNet: i) comparing our method with more advanced baselines (e.g. three different variants of recurrent neural networks) (Section 5.2); ii) testing more network architectures for ST-ResNet (Section 5.3); iii) adding the experiments of multi-step ahead prediction (Section 5.4); iv) discussing the performance of our method changing over different cloud resources (Section 5.5). Fourth, we have explored more related works (in Section 6), clarifying the differences and connections to the-state-of-the-art. This helps better position our research in the community.

2. Preliminary

We first briefly revisit the crowd flow prediction problem [3].

Definition 1 (*Region* [3]). There are many definitions of a location in terms of different granularities and semantic meanings. In this study, we partition a city into an $I \times J$ grid map based on the longitude and latitude where a grid cell denotes a region, as shown in Fig. 2(a).

Definition 2 (*Inflow/outflow* [3]). Let $\mathbb{P}$ be a collection of trajectories at the tth time interval. For a grid cell (i, j) that lies at the ith row and the jth column, the inflow and outflow of the crowds at the time interval t are defined respectively as

$$x_t^{in,i,j} = \sum_{Tr \in \mathbb{P}} |\{k > 1 | g_{k-1} \notin (i, j) \land g_k \in (i, j)\}|$$

$$x_t^{out,i,j} = \sum_{Tr \in \mathbb{P}} |\{k \geq 1 | g_k \in (i, j) \land g_{k+1} \notin (i, j)\}|$$
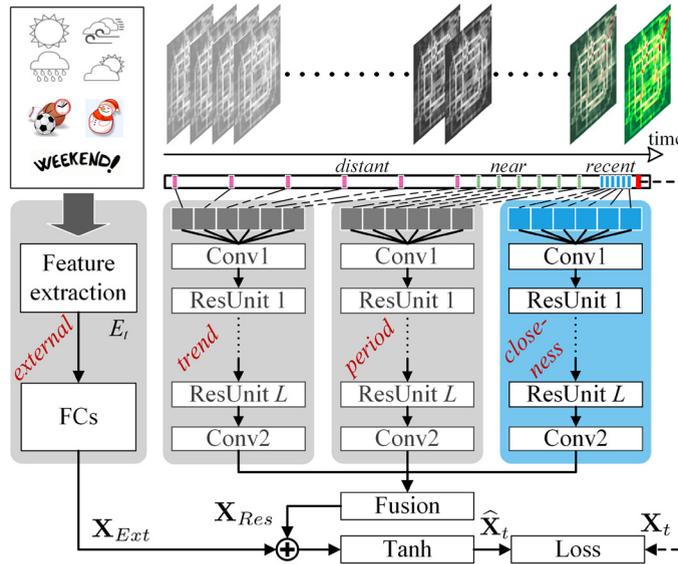
Fig. 3. ST-ResNet architecture. Conv: Convolution; ResUnit: Residual Unit; FC: Fully-connected.

where $Tr : g_1 \rightarrow g_2 \rightarrow \cdots \rightarrow g_{|Tr|}$ is a trajectory in $\mathbb{P}$, and g_k is the geospatial coordinate; $g_k \in (i, j)$ means the point g_k lies within grid (i, j), and vice versa; $|\cdot|$ denotes the cardinality of a set.

At the tth time interval, inflow and outflow in all $I \times J$ regions can be denoted as a tensor $\mathbf{X}_t \in \mathbb{R}^{2 \times I \times J}$ where $(\mathbf{X}_t)_{0,i,j} = x_t^{in,i,j}$, $(\mathbf{X}_t)_{1,i,j} = x_t^{out,i,j}$. The inflow matrix is shown in Fig. 2(b).

Formally, for a dynamical system over a spatial region represented by a $I \times J$ grid map, there are 2 types of flows in each grid over time. Thus, the observation at any time can be represented by a tensor $\mathbf{X} \in \mathbb{R}^{2 \times I \times J}$.

Problem 1. Given the historical observations $\{\mathbf{X}_t | t = 0, \cdots, n - 1\}$, predict $\mathbf{X}_n$.

3. Deep spatio-temporal residual networks

Recurrent neural networks (RNNs), like long-short term memory (LSTM), is capable of learning long-range temporal dependencies. Using RNNs, however, to model temporal *period* and *trend*, it needs very long input sequences (e.g. length = 1344),[4] which makes the whole training process non-trivial (see Section 5.2 for empirical evaluation). According to the ST domain knowledge, we know that only a few previous keyframes influence the next keyframe. Therefore, we leverage temporal *closeness*, *period*, *trend* to select keyframes for modeling. Fig. 3 presents the architecture of ST-ResNet, which is comprised of four major components modeling temporal *closeness*, *period*, *trend*, and *external* influence, respectively.

As illustrated in the top-right part of Fig. 3, we first turn inflow and outflow throughout a city at each time interval into a 2-channel image-like matrix respectively, using the approach introduced in Definitions 1 and 2. We then divide the time axis into three fragments, denoting *recent* time, *near* history and *distant* history. The 2-channel flow matrices of intervals in each time fragment are then fed into the first three components separately to model the aforementioned three temporal properties: *closeness*, *period* and *trend*, respectively. The first three components share the same network structure with a convolutional neural network followed by a Residual Unit sequence. Such structure captures the spatial dependency between nearby and distant regions. In the *external* component, we manually extract some features from external datasets, such as weather conditions and events, feeding them into a two-layer fully-connected neural network. The outputs of the first three components are fused as $\mathbf{X}_{Res}$ based on parameter matrices, which assign different weights to the results of different components in different regions. $\mathbf{X}_{Res}$ is further integrated with the output of the external component $\mathbf{X}_{Ext}$. Finally, the aggregation is mapped into $[-1, 1]$ by a Tanh function, which yields a faster convergence than the standard logistic function in the process of back-propagation learning [5] (Section 5.3.5 gives the empirical evaluation and discussion).

3.1. Structures of the first three components

The first three components (i.e. *closeness*, *period*, *trend*) share the same network structure, which is composed of two sub-components: convolution and residual unit, as shown in Fig. 4.

---

[4] Assume that half-an-hour is a time interval, 4-week sequence's length is equal to $48 \times 7 \times 4 = 1344$.

(a) Convolutions   (b) Residual Unit

Fig. 4. Convolution and residual unit.
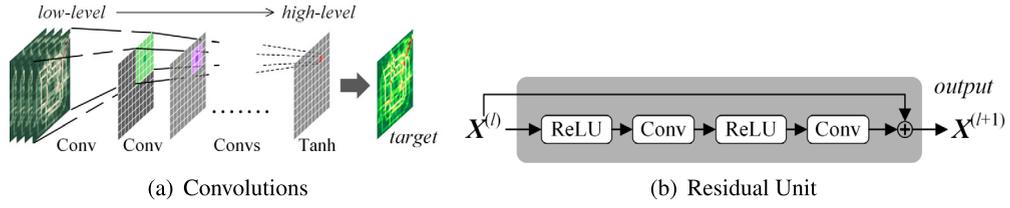
(a) *narrow* convolution   (b) *same* convolution
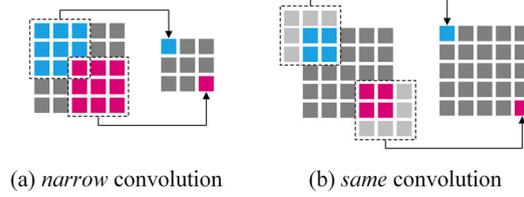
Fig. 5. Narrow and same types of convolution. The filter has size 3×3.

Convolution. A city usually has a very large size, containing many regions with different distances. Intuitively, the flow of crowds in nearby regions may affect each other, which can be effectively handled by the convolutional neural network (CNN) that has shown its powerful ability to hierarchically capture the spatial structural information [6]. In addition, subway systems and highways connect two locations with a far distance, leading to the dependency between distant regions. In order to capture the spatial dependency of any region, we need to design a CNN with many layers because one convolution only accounts for spatial near dependencies, limited by the size of their kernels. The same problem also has been found in the video sequence generating task where the input and output have the same resolution [7]. Several methods have been introduced to avoid the loss of resolution brought about by subsampling while preserving distant dependencies [8]. Being different from the classical CNN, we do not use subsampling, but only convolutions [9]. As shown in Fig. 4(a), there are three multiple levels of feature maps that are connected with a few convolutions. We find that a node in the high-level feature map depends on nine nodes of the middle-level feature map, those of which depend on all nodes in the lower-level feature map (i.e. input). It means one convolution naturally captures spatial near dependencies, and a stack of convolutions can further capture distant even citywide dependencies.

The *closeness* component of Fig. 3 adopts a few 2-channel flows matrices of intervals in the recent time to model temporal *closeness* dependency. Let the recent fragment be $[\mathbf{X}_{t-l_c}, \mathbf{X}_{t-(l_c-1)}, \cdots, \mathbf{X}_{t-1}]$, which is also known as the *closeness* dependent sequence. We first concatenate them along with the first axis (i.e. time interval) as one tensor $\mathbf{X}_c^{(0)} \in \mathbb{R}^{2l_c \times I \times J}$, which is followed by a convolution (i.e. Conv1 shown in Fig. 3) as:

$$\mathbf{X}_c^{(1)} = f\left(W_c^{(1)} * \mathbf{X}_c^{(0)} + b_c^{(1)}\right) \tag{1}$$

where $*$ denotes the convolution in a convolutional operator; f is an activation function, *e.g.* the rectifier $f(z) := \max(0, z)$ [10]; $W_c^{(1)}, b_c^{(1)}$ are the learnable parameters in the first layer.

The classical convolution has smaller output size than input size, *namely*, *narrow* convolution, as shown in Fig. 5(a). Assume that the input size is 5×5 and the filter size is 3×3 with stride 1, the output size is 3×3 if using narrow convolution. In our task, the final output size should be same as the size of the input (i.e. $I \times J$). For this goal, we employ a special type of convolution, i.e. *same* convolution (see Fig. 5(b)), which allows a filter to go outside the border of an input, padding each area outside the border with a zero.

Residual unit. It is hard to train very deep convolutional networks though the well-known activation function (e.g. ReLU) and regularization techniques are applied [11,10,12]. On the other hand, we still need a very deep network to capture very large citywide dependencies. For a typical crowd flow data, assume that the input size is 32×32, and the kernel size of convolution is fixed to 3×3, if we want to model citywide dependencies (i.e. each node in high-level layer depends on all nodes of the input), it needs more than 15 consecutive convolutional layers. To address this issue, we employ residual learning [13] in our model, which have been demonstrated to be very effective for training super deep neural networks of over 1000 layers.

In our ST-ResNet (see Fig. 3), we stack L residual units upon Conv1 as follows,

$$\mathbf{X}_c^{(l+1)} = \mathbf{X}_c^{(l)} + \mathcal{F}(\mathbf{X}_c^{(l)}; \theta_c^{(l)}), l = 1, \cdots, L \tag{2}$$

where $\mathcal{F}$ is the residual function (i.e. two combinations of "ReLU + Convolution", see Fig. 4(b)), and $\theta^{(l)}$ includes all learnable parameters in the lth residual unit. We also attempt *Batch Normalization* (BN) [11] that is added before ReLU. On top of the Lth residual unit, we append a convolutional layer (i.e. Conv2 shown in Fig. 3). With 2 convolutions and L residual units, the output of the *closeness* component of Fig. 3 is $\mathbf{X}_c^{(L+2)}$.

(a) Feb 8-14 (red), Feb 15-21 (green), 2016   (b) Aug 10-12 (red), Aug 17-19 (green), 2013
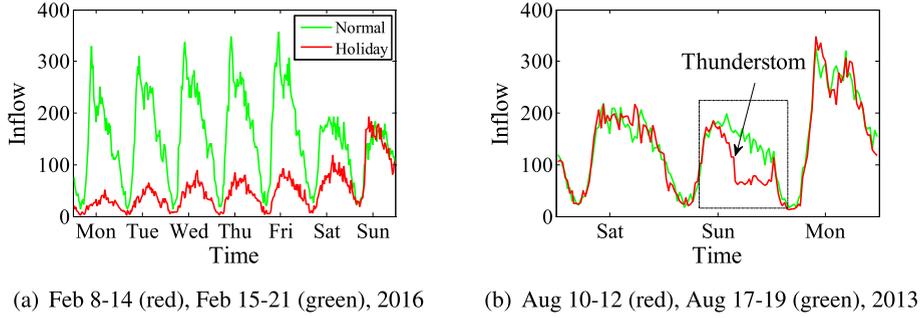
Fig. 6. Effects of holidays and weather in Office Area of Beijing (the region is shown in Fig. 2(a)).

Likewise, using the above operations, we can construct the *period* and *trend* components of Fig. 3. Assume that there are l_p time intervals from the period fragment and the period is p. Therefore, the *period* dependent sequence is $[\mathbf{X}_{t-l_p \cdot p}, \mathbf{X}_{t-(l_p-1)\cdot p}, \cdots, \mathbf{X}_{t-p}]$. With the convolutional operation and L residual units like in Eqs. (1) and (2), the output of the *period* component is $\mathbf{X}_p^{(L+2)}$. Meanwhile, the output of the *trend* component is $\mathbf{X}_q^{(L+2)}$ with the input $[\mathbf{X}_{t-l_q \cdot q}, \mathbf{X}_{t-(l_q-1)\cdot q}, \cdots, \mathbf{X}_{t-q}]$ where l_q is the length of the *trend* dependent sequence and q is the trend span. Note that p and q are actually two different types of periods. In the detailed implementation, p is equal to one-day that describes daily periodicity, and q is equal to one-week that reveals the weekly trend.

3.2. The structure of the external component

Traffic flows can be affected by many complex external factors, such as weather and events. Fig. 6(a) shows that crowd flows during holidays (Chinese Spring Festival) can be significantly different from the flows during normal days. Fig. 6(b) shows that heavy rain sharply reduces the crowd flows at Office Area compared to the same day of the latter week. Let E_t be the feature vector that represents these external factors at predicted time interval t. In our implementation, we mainly consider weather, holiday event, and metadata (e.g. DayOfWeek, Weekday/Weekend). The details are introduced in Table 2. To predict flows at time interval t, the holiday event and metadata can be directly obtained. However, the weather at future time interval t is unknown. Instead, one can use the forecasting weather at time interval t or the approximate weather at time interval $t-1$. Formally, we stack two fully-connected layers upon E_t, the first layer can be viewed as an embedding layer for each sub-factor followed by an activation. The second layer is used to map low to high dimensions that have the same shape as $\mathbf{X}_t$. The output of the *external* component of Fig. 3 is denoted as $\mathbf{X}_{Ext}$ with the parameters θ_{Ext}.

3.3. Fusion

In this section, we discuss how to fuse four components of Fig. 3. We firstly fuse the *closeness*, *period*, and *trend* components with a parametric-matrix-based fusion method, which is then further combined with the *external* component.

Figs. 7(a) and (d) show the ratio curves using Beijing trajectory data presented in Table 2 where x-axis is time gap between two time intervals and y-axis is the average ratio value between arbitrary two inflows that have the same time gap. The curves from two different regions all show an empirical temporal correlation in time series, namely, inflows of recent time intervals are more relevant than ones of distant time intervals, which implies temporal *closeness*. The two curves have different shapes, which demonstrates that different regions may have different characteristics of closeness. Figs. 7(b) and (e) depict inflows at all time intervals of 7 days. We can see the obvious *daily periodicity* in both regions. In Office Area, the peak values on weekdays are much higher than ones on weekends. Residential Area has similar peak values for both weekdays and weekends. Figs. 7(c) and (f) describe inflows at a certain time interval (9:00pm–9:30pm) of Tuesday from March 2015 and June 2015. As time goes by, the inflow progressively decreases in Office Area, and increases in Residential Area. It shows the different trends in different regions. In summary, inflows of two regions are all affected by *closeness*, *period*, and *trend*, but the degrees of influence may be very different. We also find the same properties in other regions as well as their outflows.

Above all, the different regions are all affected by *closeness*, *period* and *trend*, but the degrees of influence may be different. Inspired by these observations, we propose a parametric-matrix-based fusion method.

Parametric-matrix-based fusion. We fuse the first three components (i.e. *closeness*, *period*, *trend*) of Fig. 3 as follows

$$\mathbf{X}_{Res} = \mathbf{W}_c \circ \mathbf{X}_c^{(L+2)} + \mathbf{W}_p \circ \mathbf{X}_p^{(L+2)} + \mathbf{W}_q \circ \mathbf{X}_q^{(L+2)} \in \mathbb{R}^{2 \times I \times J} \tag{3}$$

or, equivalently,

$$x_{Res}^{k,i,j} = w_c^{k,i,j} \cdot x_c^{k,i,j} + w_p^{k,i,j} \cdot x_p^{k,i,j} + w_q^{k,i,j} \cdot x_q^{k,i,j}, \forall k \in \{0,1\}, \forall i \in \{0, \cdots, I-1\}, \forall j \in \{0, \cdots, J-1\} \tag{4}$$
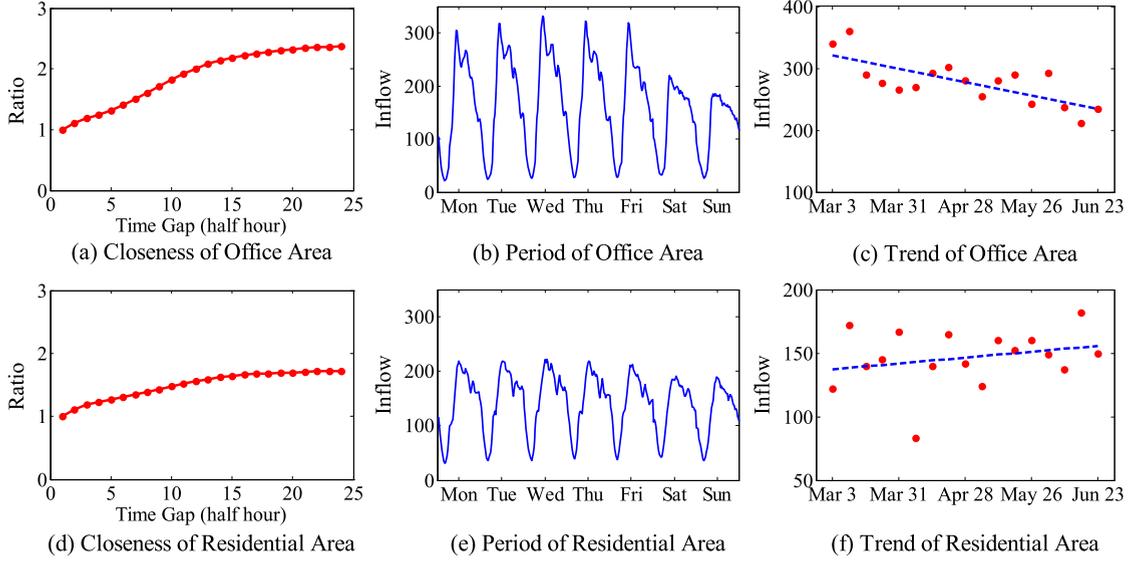
Fig. 7. Temporal dependencies (Office Area and Residential Area are shown in Fig. 2(a)).

where ∘ is Hadamard product (i.e. element-wise multiplication), $w_c^{k,i,j} \in \mathbf{W}_c$, $w_p^{k,i,j} \in \mathbf{W}_p$ and $w_q^{k,i,j} \in \mathbf{W}_q$ are the learnable parameters that adjust the degrees affected by closeness, period and trend, respectively. $x_{Res}^{k,i,j}, x_c^{k,i,j}, x_p^{k,i,j}$, and $x_q^{k,i,j}$ are respectively an entry of $\mathbf{X}_{Res}, \mathbf{X}_c^{(L+2)}, \mathbf{X}_p^{(L+2)}$, and $\mathbf{X}_q^{(L+2)}$.

Fusing the external component. We here directly merge the output of the first three components with that of the *external* component, as shown in Fig. 3. Finally, the predicted value at the *t*th time interval, denoted by $\widehat{\mathbf{X}}_t$, is defined as

$$\widehat{\mathbf{X}}_t = \tanh(\mathbf{X}_{Res} + \mathbf{X}_{Ext}) \tag{5}$$

where tanh is a hyperbolic tangent that ensures the output values are between −1 and 1.

Our ST-ResNet can be trained to predict $\mathbf{X}_t$ from three sequences of flow matrices and external factor features by minimizing mean squared error between the predicted flow matrix and the true flow matrix:

$$\mathcal{L}(\theta) = \|\mathbf{X}_t - \widehat{\mathbf{X}}_t\|_2^2 \tag{6}$$

where θ are all learnable parameters in the ST-ResNet.

3.4. Algorithms and optimization

Algorithm 1 outlines the ST-ResNet training process. We first construct the training instances from the original sequence data (lines 1–6). Then, ST-ResNet is trained via backpropagation and Adam [14] (lines 7–11).

After training, the learned ST-ResNet model $\mathcal{M}$ is obtained for the single- or multi-step look-ahead prediction. the process of which is summarized in Algorithm 2. Some types of external features (i.e. weather) used here are different from that in Algorithm 1. In the training process, we use the true weather data, which is replaced by the forecasted weather data in Algorithm 2.

3.5. Discussion on different approaches to construct inputs

We here present two different approaches to construct inputs of *closeness*, *period*, and *trend* components of Fig. 3. Fig. 8 gives an illustration. Let period (*p*) and trend span (*q*) be 1 day and 1 week, respectively. According to three temporal properties (i.e. closeness, period, trend), the original constructing approach (Fig. 8(a)) selects the following key frames:

- dependent inputs of *closeness* component: $[\mathbf{X}_{t-l_c}, \mathbf{X}_{t-(l_c-1)}, \cdots, \mathbf{X}_{t-1}]$.
- dependent inputs of *period* component: $[\mathbf{X}_{t-l_p \cdot p}, \mathbf{X}_{t-(l_p-1) \cdot p}, \cdots, \mathbf{X}_{t-p}]$.
- dependent inputs of *trend* component: $[\mathbf{X}_{t-l_q \cdot q}, \mathbf{X}_{t-(l_q-1) \cdot q}, \cdots, \mathbf{X}_{t-q}]$.

Based on the original approach, we here introduce two approaches (AVG1 and AVG2) that employ averaging the historical frames, in which dependent inputs of *closeness* component are same with that of the original approach.

AVG1 (Fig. 8(b)): averaging respectively dependent inputs of *period* and *trend* components of the **original** approach as new dependent inputs of *period* and *trend* components, showing below:

Algorithm 1: Training of ST-ResNet.

Input: Historical observations: $\{\mathbf{X}_0, \cdots, \mathbf{X}_{n-1}\}$;
      external features: $\{E_0, \cdots, E_{n-1}\}$;
      lengths of *closeness, period, trend* sequences: l_c, l_p, l_q;
      period: p; trend span: q.
Output: ST-ResNet model $\mathcal{M}$.
  // construct training instances
1 $\mathcal{D} \longleftarrow \emptyset$
2 **for** *all available time interval* $t(1 \le t \le n-1)$ **do**
3      $\mathcal{S}_c = [\mathbf{X}_{t-l_c}, \mathbf{X}_{t-(l_c-1)}, \cdots, \mathbf{X}_{t-1}]$
4      $\mathcal{S}_p = [\mathbf{X}_{t-l_p \cdot p}, \mathbf{X}_{t-(l_p-1)\cdot p}, \cdots, \mathbf{X}_{t-p}]$
5      $\mathcal{S}_q = [\mathbf{X}_{t-l_q \cdot q}, \mathbf{X}_{t-(l_q-1)\cdot q}, \cdots, \mathbf{X}_{t-q}]$
      // $\mathbf{X}_t$ is the target at time t
6   put an training instance $(\{\mathcal{S}_c, \mathcal{S}_p, \mathcal{S}_q, E_t\}, \mathbf{X}_t)$ into $\mathcal{D}$

  // train the model
7 initialize the parameters θ
8 **repeat**
9   randomly select a batch of instances $\mathcal{D}_b$ from $\mathcal{D}$
10   find θ by minimizing the objective (6) with $\mathcal{D}_b$
11 **until** *stopping criteria is met*
12 output the learned ST-ResNet model $\mathcal{M}$

Algorithm 2: Multi-step Ahead Prediction Using ST-ResNet.

Input: Learned ST-ResNet model: $\mathcal{M}$;
     number of look-ahead steps: k;
     historical observations: $\{\mathbf{X}_0, \cdots, \mathbf{X}_{n-1}\}$;
     external features: $\{E_n, \cdots, E_{n+k-1}\}$;
     lengths of *closeness, period, trend* sequences: l_c, l_p, l_q;
     period: p; trend span: q.
1 $\mathcal{X} \longleftarrow \{\mathbf{X}_0, \cdots, \mathbf{X}_{n-1}\}$ *// (i.e. $\mathcal{X}_t = \mathbf{X}_t, \forall t$)*
2 **for** $t = n$ **to** $n+k-1$ **do**
3      $\mathcal{S}_c = [\mathcal{X}_{t-l_c}, \mathcal{X}_{t-(l_c-1)}, \cdots, \mathcal{X}_{t-1}]$
4      $\mathcal{S}_p = [\mathcal{X}_{t-l_p \cdot p}, \mathcal{X}_{t-(l_p-1)\cdot p}, \cdots, \mathcal{X}_{t-p}]$
5      $\mathcal{S}_q = [\mathcal{X}_{t-l_q \cdot q}, \mathcal{X}_{t-(l_q-1)\cdot q}, \cdots, \mathcal{X}_{t-q}]$
6      $\widehat{\mathbf{X}}_t \longleftarrow \mathcal{M}(\mathcal{S}_c, \mathcal{S}_p, \mathcal{S}_q, E_t)$
7   put $\widehat{\mathbf{X}}_t$ into $\mathcal{X}$, i.e. $\mathcal{X}_t = \widehat{\mathbf{X}}_t$
8 output $\{\widehat{\mathbf{X}}_n, \cdots, \widehat{\mathbf{X}}_{n+k-1}\}$

- dependent input of *period* component of AVG1: $\frac{1}{l_p} \sum_{i=1}^{l_p} \mathbf{X}_{t-i \cdot p}$.

- dependent input of *trend* component of AVG1: $\frac{1}{l_q} \sum_{i=1}^{l_q} \mathbf{X}_{t-i \cdot q}$.

Compared to the original approach, AVG1 has the same raw inputs, but requires a model that has fewer trainable parameters because its *period* and *trend* components both has 1 frame.

AVG2: averaging the frames around same time intervals in the original approach, producing the same number frames as the original approach (see Fig. 8(c) for example). Formally, their inputs are below:

- dependent inputs of *period* component of AVG2: $[\overline{\mathbf{X}}_{t-l_p \cdot p}, \overline{\mathbf{X}}_{t-(l_p-1)\cdot p}, \cdots, \overline{\mathbf{X}}_{t-p}]$, where $\overline{\mathbf{X}}_{t-j \cdot p} = \frac{1}{3}(\mathbf{X}_{t-j \cdot p-1} + \mathbf{X}_{t-j \cdot p} + \mathbf{X}_{t-j \cdot p+1})$, $j = 1, \cdots l_p$.

- dependent inputs of *trend* component AVG2: $[\overline{\mathbf{X}}_{t-l_q \cdot q}, \overline{\mathbf{X}}_{t-(l_q-1)\cdot q}, \cdots, \overline{\mathbf{X}}_{t-q}]$. $\forall j \in \{1, \cdots, l_q\}$, $\overline{\mathbf{X}}_{t-j \cdot q} = \frac{1}{5} \sum_{X_k \in X_{j_{weekdays}}} \mathbf{X}_k$ if X_t is in weekday or $\overline{\mathbf{X}}_{t-j \cdot q} = \frac{1}{2} \sum_{X_k \in X_{j_{weekend}}} \mathbf{X}_k$ otherwise, where $X_{j_{weekdays}}$ denotes 5 weekdays and $X_{j_{weekend}}$ denotes weekend in the previous jth trend span.

Compared to the original approach, AVG2 has the same trainable parameters, but capable of capturing more raw frames.

4. System architecture

Fig. 9 presents the framework of our system, which consists of three major parts: *local GPU servers*, and the *Cloud*, and *users* (e.g. website and QR Code), resulting in offline and online data flows, respectively. The local GPU servers store historical observations, such as taxi trajectories, meteorological data. The Cloud receive real-time data, including real-time traffic data
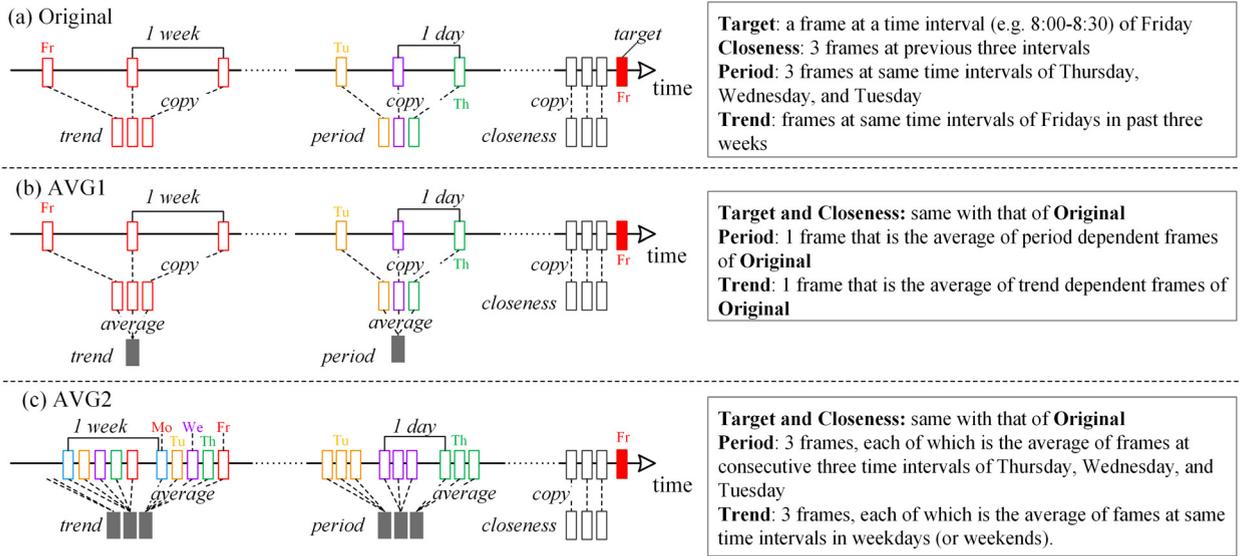
Fig. 8. Input constructing approaches. Mo: Monday; Tu: Tuesday; We: Wednesday; Th: Thursday; Fr: Friday. Different colors denote different weekdays. (Best viewed in color).
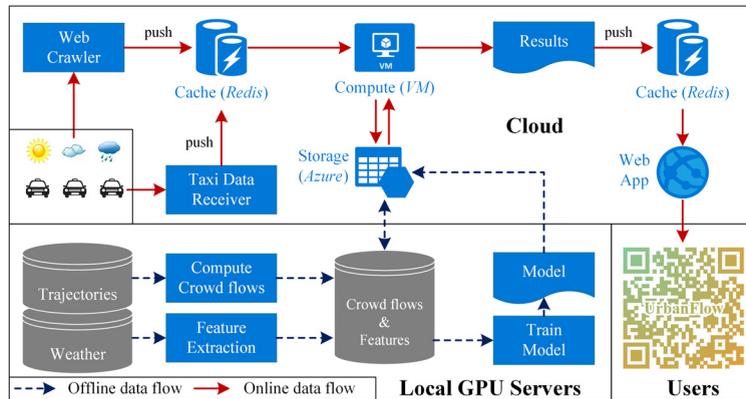
Fig. 9. System Framework.

(e.g. trajectories) within a time interval as well as meteorological data. The users access the inflow/outflow data, displaying them on websites or smart phone via scanning QR code.

4.1. The cloud

The cloud continuously receives GPS trajectories of taxicabs and crawls meteorological data, and then caches them into a *Redis*.[5] A virtual machine (VM) (or VMs) on the Cloud pulls these data from the *Redis*, and then computes crowd flows according to GPS trajectories for each and every region of a city. Meanwhile, the VM extracts the features from meteorological data, event data and others. Afterwards, the VM stores the crowd flow data and extracted features into the *storage* (a part of the VM). To save the resource on the cloud (more storages need more expensive payment), we only store the crowd flow data and features in past two days. Historical data can be moved to local servers periodically.

We use Azure platform as a service (PaaS). Table 1 details the Azure[6] resources for our system. We employ a VM,[7] called *A2 standard in Azure*, that has 2 cores and 3.5 GB memory for forecasting the crowd flows in near future. The website and web service share a *App Service*, given the potential heavy accesses by many users. As the historical data is stored in local servers, a 6 GB *Redis Cache* is enough for caching the real-time trajectories in past half-hour, crowd flow data and extracted features in past two days, and inferred results.

---

[5] Redis is an in-memory data structure store, used as a database (https://redis.io/).

[6] https://azure.microsoft.com.

[7] To accelerate the computation, one can choose a more powerful VM, such as *D4 standard* of Azure that has 8 cores and 28 GB memory.

Table 1
The Azure resources used for our system.

Azure Service	Configuration
App Service	Standard, 4 cores, 7 GB memory
Virtual Machine	A2 standard, 2 cores, 3.5 GB memory
Redis Cache	P1 premium, 6 GB

4.2. Local GPU servers

Although all the jobs can be done on the cloud, GPU services on the Cloud is not supported in some areas (e.g. China). On the other hand, we need to pay for other cloud services, like storages and I/O bandwidths. Saving unnecessary cost is vital for a research prototype. In addition, migrating massive data from local servers up to the cloud is time-consuming given the limited network bandwidth. For instance, the historical trajectories can be hundreds of Gigabytes, even Terabytes, leading to a very long time for copying the data from local servers to the cloud.

Therefore, we employ a hybrid framework that combines local GPU servers with the cloud. *Local GPU servers* mainly handle the offline training (learning), including three tasks:

- Converting trajectories into inflow/outflow: we first use the massive historical trajectories and employ a calculation module to get crowd flow data, then store them in local.
- Extracting features from external data: we fist collect external data (e.g. weather, holiday events) from different data sources and fit them into a feature extraction module to get continuous or discrete features, and then store them in local.
- Training model: we use above generated crowd flows and external features to train a predictive model via our proposed ST-ResNet, and then upload the learned model to the cloud. Note that as the dynamic crowd flows and features are stored in the Storage (Azure), we sync up the online data to local servers before each training processing. In this way, we are agile to try new ideas (e.g. re-train the model) while greatly reducing expense for a research prototype.

4.3. User interface

Fig. 10(a) presents the website of UrbanFlow [15], where each grid on the map stands for a region and the number associated with it denotes its inflow or outflow of crowds. The user can view inflow or outflow via the top-right button named "InFlow/OutFlow". The smaller the number is, the sparser the crowd flow. The color of each grid is determined in accordance with its crowd flows, e.g. "red" means "dense" crowd flow and "green" means "sparse" crowd flow. The top-right corner of the website shows the buttons which can switch between different types of flows. A user can select any grid (representing a region) on the website and click it to see the region's detailed flows, as shown in Fig. 10(b) where blue, black, and green curves indicate flows of yesterday, past, and future times at today, respectively. The bottom of the website shows a few sequential timestamps. The heatmap at a certain timestamp will be shown in the website when a user clicks the associated timestamp. Intuitively, the user can watch the movie-style heatmaps (Fig. 10(c)) by clicking "play button" at the bottom-left of Fig. 10(a). At present, we apply UrbanFlow to the area of Guiyang City, China.[8]

5. Experiments

In this section, we evaluate our ST-ResNet on two types of crowd flows in Beijing and NYC against 9 baselines.

5.1. Settings

Datasets. We use two different sets of data as shown in Table 2. Each dataset contains two sub-datasets: trajectories and weather, as detailed as follows.

- **TaxiBJ**: Trajectory data is the taxicab GPS data and meteorology data in Beijing from four time intervals: 1st Jul. 2013–30th Otc. 2013, 1st Mar. 2014–30th Jun. 2014, 1st Mar. 2015–30th Jun. 2015, 1st Nov. 2015–10th Apr. 2016. Using Definition 2, we obtain two types of crowd flows. We choose data from the last four weeks as the testing data, and all data before that as training data.
- **BikeNYC**: Trajectory data is taken from the NYC Bike system in 2014, from Apr. 1st to Sept. 30th. Trip data includes: trip duration, starting and ending station IDs, and start and end times. Among the data, the last 10 days are chosen as testing data, and the others as training data.

Baselines. The baselines and our models are outlined as follows.

(a) Web user interface of UrbanFlow

(b) Inflow in a certain region

(c) Heatmaps in past & future
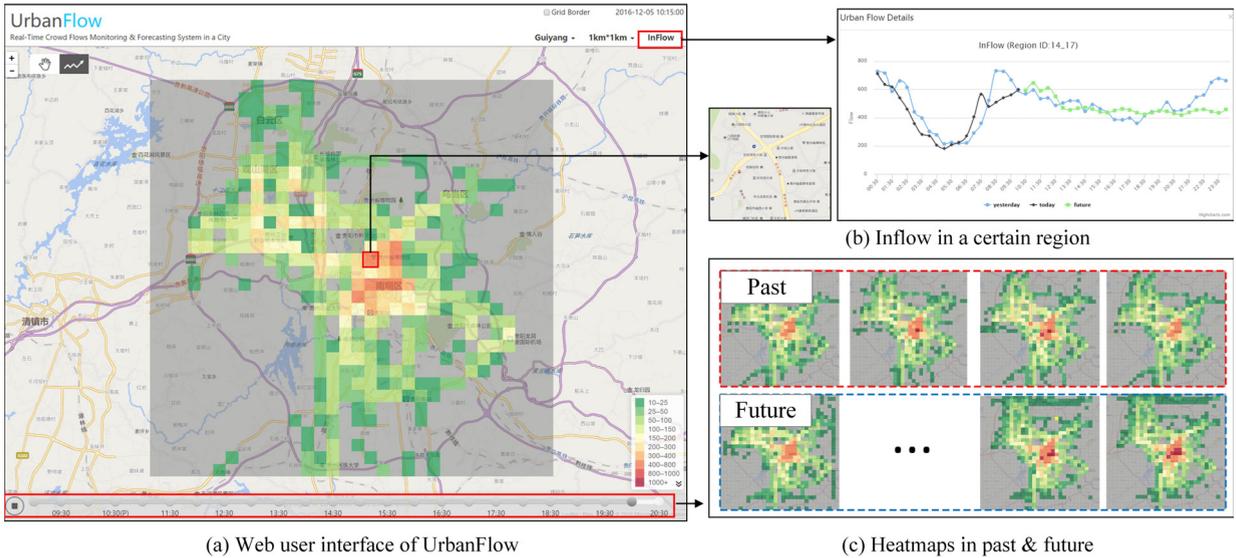
Fig. 10. Web user interface of UrbanFlow.

Table 2
Datasets (holidays include adjacent weekends).

Dataset	TaxiBJ	BikeNYC
Data type	Taxi GPS	Bike rent
Location	Beijing	New York
Time Span	7/1/2013–10/30/2013 3/1/2014–6/30/2014 3/1/2015–6/30/2015 11/1/2015–4/10/2016	4/1/2014–9/30/2014
Time interval	30 minutes	1 hour
Gird map size	(32, 32)	(16, 8)
Trajectory data		
Average sampling rate (s)	∼60	\
# taxis/bikes	34,000+	6,800+
# available time interval	22,459	4,392
External factors (holidays and meteorology)		
# holidays	41	20
Weather conditions	16 types (e.g. Sunny, Rainy)	\
Temperature/°C	[−24.6, 41.0]	\
Wind speed/mph	[0, 48.6]	\

- **HA**: We predict inflow and outflow of crowds by the average value of historical inflow and outflow in the corresponding periods, e.g. 9:00am–9:30am on Tuesday, its corresponding periods are all historical time intervals from 9:00am to 9:30am on all historical Tuesdays.
- **ARIMA**: Auto-Regressive Integrated Moving Average (ARIMA) is a well-known model for understanding and predicting future values in a time series.
- **SARIMA**: Seasonal ARIMA. Beyond ARIMA, SARIMA also considers the seasonal terms, capable of both learning closeness and periodic dependencies.
- **VAR**: Vector Auto-Regressive (VAR) is a more advanced spatio-temporal model, which can capture the pairwise relationships among all flows, and has heavy computational costs due to the large number of parameters.
- **ST-ANN**: It first extracts spatial (nearby 8 regions' values) and temporal (8 previous time intervals) features, then fed into an artificial neural network.
- **DeepST** [3]: a deep neural network (DNN)-based prediction model for spatio-temporal data, which shows state-of-the-art results on the crowd flow prediction.
- **RNN** [37]: recurrent neural network (RNN), a deep learning model, which can capture temporal dependencies. Formally, RNN can train on sequences with the arbitrary length. In our experiment, we fix the length of input sequence as one of {3, 6, 12, 24, 48, 336}. Taking 48 as example, the dependent input sequence is just a one-day data if the interval time

Table 3
Characteristics of baselines. Im: implicit; Ex: explicit; ×: not include; ✓: include.

Model	Spatial		Temporal			External factors
	Intra	Inter	Closeness	Period (daily)	Trend (weekly)	
HA	×	×	×	Im	Im	×
ARIMA	×	×	Ex	×	×	×
SARIMA	×	×	Ex	Ex	×	×
VAR	×	✓	Ex	×	×	×
ST-ANN	×	✓	Ex	×	×	×
DeepST	✓	✓	Ex	Ex	Ex	✓
RNN-3/6/12/24	×	×	Ex	×	×	×
RNN-48	×	×	Ex	Im	×	×
RNN-336	×	×	Ex	Im	Im	×
ST-ResNet	✓	✓	Ex	Ex	Ex	✓

Table 4
Details of convolutions and residual units.

Layer name	Output size	Closeness	Period	Trend
Conv1	32×32	$3 \times 3, 64$	$3 \times 3, 64$	$3 \times 3, 64$
ResUnit 1	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
ResUnit 2	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
ResUnit 3	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
ResUnit 4	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
Conv2	32×32	$3 \times 3, 2$	$3 \times 3, 2$	$3 \times 3, 2$

is equal to 30 minutes. Therefore, we have 6 RNN variants, including RNN-3, RNN-6, RNN-12, RNN-24, RNN-48, and RNN-336.

- **LSTM** [16]: Long-short-term-memory network (LSTM), a special kind of RNN, capable of learning long-term temporal dependencies. Being same as the setting of RNN, we conduct the experiments on 6 LSTM variants, i.e. LSTM-3, LSTM-6, LSTM-12, LSTM-24, LSTM-48, and LSTM-336.
- **GRU** [17]: Gated-recurrent-unit network, a new kind of RNN, can be used to capture long-term temporal dependencies. Being same as the setting of RNN, the following GRU variants are selected as the baselines: GRU-3, GRU-6, GRU-12, GRU-24, GRU-48, and GRU-336.
- **ST-ResNet**: The model proposed in this paper. The number of residual units is set as 12 for the dataset TaxiBJ, and 4 for BikeNYC.
- **ST-ResNet-AVG1** and **ST-ResNet-AVG2**: two ST-ResNet variants with same setting, but different input constructing approaches (see Section 3.5)

Table 3 shows characteristics of these baselines. LSTM and GRU models have the same characteristics as that of RNN. For example, the HA model implicitly capture daily period and weekly trend, but cannot capture spatial dependencies or external factors; VAR considers the relations between inflow and outflow within a region; ST-ANN leverage the flows nearby regions; RNN-336 have assess to daily period and weekly trends, but they are not explicit. Our ST-ResNet can capture all spatial and temporal dependencies as well as external factors.

Preprocessing. In the output of the ST-ResNet, we use tanh as our final activation (see Eq. (5)), whose range is between -1 and 1. Here, we use the Min-Max normalization method to scale the data into the range $[-1, 1]$. In the evaluation, we re-scale the predicted value back to the normal values, compared with the groundtruth. For external factors, we use one-hot coding to transform metadata (i.e. DayOfWeek, Weekend/Weekday), holidays and weather conditions into binary vectors, and use Min-Max normalization to scale the Temperature and Wind speed into the range $[0, 1]$.

Hyperparameters. The learnable parameters are initialized using a uniform distribution with the default parameter in Keras [18]. The convolutions of Conv1 and all residual units use 64 filters of size 3×3, and Conv2 uses a convolution with 2 filters of size 3×3. For example, a 4-residual-unit of ST-ResNet consists of Conv1, 4 residual unit, and Conv2. See Table 4 for the details. The Adam [14] is used for optimization, and the batch size is 32. There are 5 extra hyperparameters in our ST-ResNet, of which p and q are empirically fixed to one-day and one-week, respectively. For lengths of the three dependent sequences, we set them as: $l_c \in \{1, 2, 3, 4, 5\}, l_p \in \{1, 2, 3, 4\}, l_q \in \{1, 2, 3, 4\}$. We select 90% of the training data for training each model, and the remaining 10% is chosen as the validation set, which is used to early-stop our training algorithm for each model based on the best validation score. Afterwards, we continue to train the model on the full training data for a fixed number of epochs (e.g. 10, 100 epochs).

Table 5
Experimental environment.

OS	Ubuntu 16.04
Memory	512 GB
CPU	Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60 GHz
GPU	Tesla P100
CUDA version	8.0
cuDNN version	8.0
Keras version	2.0
TensorFlow version	1.3

Evaluation metric: We measure our method by Root Mean Square Error (RMSE)[9] as

$$RMSE = \sqrt{\frac{1}{z} \sum_i (x_i - \hat{x}_i)^2} \tag{7}$$

where x and $\hat{x}$ are the available ground truth and the corresponding predicted value, respectively; z is the number of all available ground truths.

Experiments are mainly run on a GPU server, whose detailed information is shown in Table 5. The python libraries, including TensorFlow [19] and Keras [18], are used to build our models.

5.2. Evaluation of single-step ahead prediction

In this section, we evaluate the single-step ahead prediction, *namely*, predicting the crowd flows at time t using the historical observations. Table 6 shows the RMSE of all methods on both TaxiBJ and BikeNYC. Our ST-ResNet consistently and significantly outperforms all baselines. Specifically, the results on TaxiBJ demonstrates that ST-ResNet (with 12 residual units) is relatively 25% better than ARIMA, 36% better than SARIMA, 25% better than VAR, 12% better than ST-ANN, 6% better than DeepST, 36% to 66% better than RNN, 34% to 58% better than LSTM, 25% to 57% better than GRU. ST-ResNet-noExt is a degraded version of ST-ResNet that does not consider the *external* factors (e.g. meteorology data). We can see that ST-ResNet-noExt is slightly worse than ST-ResNet, pointing out external factors are always beneficial. DeepST exploits spatio-temporal CNNs and is clearly better than other baselines. While both ST-ANN and VAR use spatial/temporal information and relationships among flows, they are worse than DeepST because they only consider the *near* spatial information and *recent* temporal information. Among the temporal models, GRU and LSTM have similar RMSE, and outperform RNN in average because GRU and LSTM both can capture long-term temporal dependencies. However, GRU-336 and LSTM-336 have very bad performance as well as RNN-336, which demonstrates RNN-based models cannot capture very long-term dependencies (i.e. *period* and *trend*). Intuitively, we rank all of these models, as shown in Fig. 11(a).

Being different from TaxiBJ, BikeNYC consists of two different types of crowd flows, including new-flow and end-flow [20]. We here adopt a total of 4-residual-unit ST-ResNet, and consider the metadata as external features like DeepST [3]. ST-ResNet has relatively from 15% up to 71% lower RMSE than these baselines, demonstrating that our proposed model has good generalization performance on other flow prediction tasks. Fig. 11(b) depicts the ranking of these models.

For different input constructing approaches (AVG1 and AVG2 in Section 3.5), we observe ST-ResNet-AVG1 is worst in both datasets because it averages the period dependents and trend dependents, though the model complexity reduces, its error raises. ST-ResNet-AVG2 is worse than ST-ResNet in TaxiBJ but better in BikeNYC. The reason may be that TaxiBJ has some missing time intervals in the data, but there are no missing in BikeNYC, which means ST-ResNet has more training instances than ST-ResNet-AVG2 in the incomplete dataset.

5.3. Results of different ST-resnet variants

We here present the results of different ST-ResNet variants, including changing network configurations, network depth, and different components used.

5.3.1. Impact of different configurations

Fig. 12 shows the results of different configurations. The same hyper-parameters: $l_c = 3$, $l_p = 1$, $l_q = 1$, *number of residual unit* = 12.

- *Effect of batch normalization (BN)*: We attempt to adopt BN into each residual unit, finding that the RMSE slightly improves in single-step ahead prediction, as shown in Fig. 12(a).

---

[9] The smaller the better.

Table 6
Comparisons with baselines on TaxiBJ and BikeNYC. Results of ARIMA, SARIMA, VAR and DeepST on BikeNYC are taken from [3]. For RNN-based models and ST-ResNet variants, we run each of them 10 times and show "mean ± standard deviation".

Model	RMSE	
	TaxiBY	BikeNYC
HA	57.69	21.578
ARIMA	22.78	10.07
SARIMA	26.88	10.56
VAR	22.88	9.92
ST-ANN	19.57	7.58
DeepST	18.18	7.43
RNN-3	26.68 ± 3.41	9.01 ± 0.50
RNN-6	30.03 ± 1.60	8.61 ± 0.42
RNN-12	45.51 ± 2.01	12.41 ± 0.84
RNN-24	51.12 ± 1.99	12.53 ± 0.14
RNN-48	43.42 ± 1.20	12.76 ± 0.18
RNN-336	39.61 ± 0.77	10.74 ± 0.35
LSTM-3	26.81 ± 2.80	8.67 ± 0.33
LSTM-6	26.07 ± 1.87	9.56 ± 0.64
LSTM-12	27.59 ± 3.69	9.90 ± 0.67
LSTM-24	25.69 ± 2.25	11.34 ± 0.44
LSTM-48	27.80 ± 2.87	12.36 ± 0.87
LSTM-336	40.68 ± 1.08	10.54 ± 0.13
GRU-3	22.97 ± 1.11	8.76 ± 0.37
GRU-6	23.64 ± 1.14	8.57 ± 0.22
GRU-12	27.40 ± 3.72	9.68 ± 0.51
GRU-24	27.01 ± 1.58	12.27 ± 0.77
GRU-48	28.56 ± 3.71	12.71 ± 0.93
GRU-336	40.27 ± 2.30	10.76 ± 0.33
ST-ResNet	$\mathbf{17.17 \pm 0.21}$ (12 residual units)	6.32 ± 0.13 (4 residual units)
ST-ResNet-noExt	17.26 ± 0.38 (12 residual units)	\
ST-ResNet-AVG1	19.07 ± 0.30 (12 residual units)	6.44 ± 0.19 (4 residual units)
ST-ResNet-AVG2	18.04 ± 0.15 (12 residual units)	$\mathbf{5.99 \pm 0.09}$ (4 residual units)
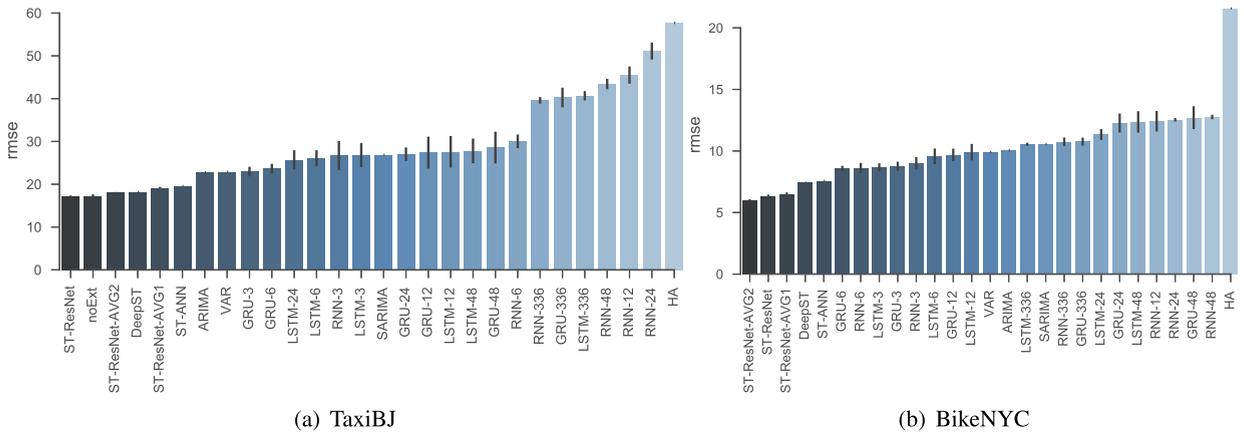
(a) TaxiBJ

(b) BikeNYC

Fig. 11. Model ranking on TaxiBJ and BikeNYC using RMSE (mean). The smaller the better. For deep learning models, we use error bars to show standard deviation.

(a) effect of BN

(b) different fusion mechanisms

(c) different residual units

(d) different weather data
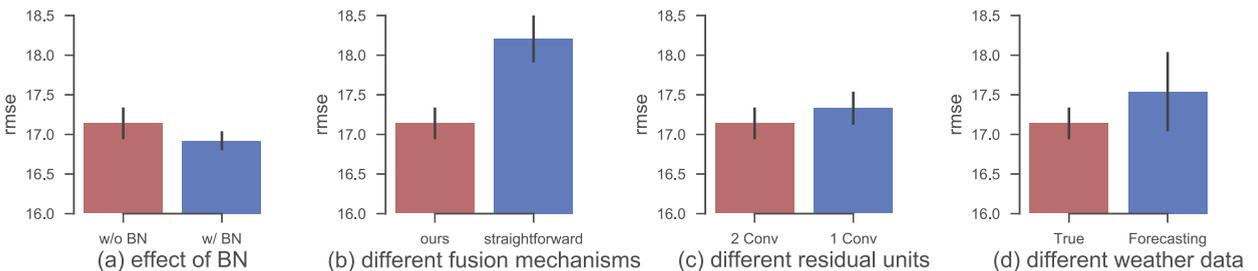
Fig. 12. Results of different configurations. We run each configuration 10 times where error bars show standard deviation.
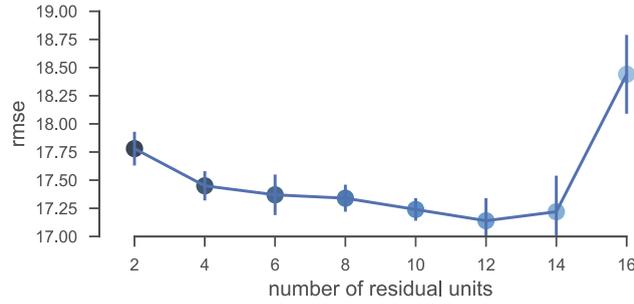
Fig. 13. Impact of network depth. We run each model 10 times where error bars show standard deviation.

(a) filter size
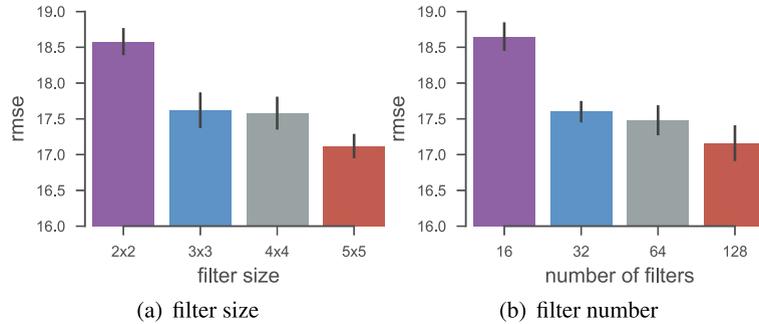
(b) filter number

Fig. 14. Impact of filter size and number. We run each model 10 times where error bars show standard deviation.

- *Effect of parametric-matrix-based fusion*: We use a parametric-matrix-based fusion mechanism (see Eq. (3)) to fuse temporal *closeness*, *period* and *trend* components. Simply, one also can employ a straightforward method for fusing, i.e. $\mathbf{X}_c^{(L+2)} + \mathbf{X}_p^{(L+2)} + \mathbf{X}_q^{(L+2)}$. Fig. 12(b) shows that ours is better than the straightforward method, demonstrating the effectiveness of our proposed parametric-matrix-based fusion.
- *Internal structure of residual unit*: The proposed residual unit includes 2 convolutions. We here test the performance different setting in the residual unit. From Fig. 12(c), we observe that the model using 2 convolutions is better than using 1 convolution.
- *True Weather vs. Forecast*: Using the same network architecture, we here employ different weather data as the training data: true weather and forecasting. Fig. 12(d) shows the comparison of different weather, demonstrating that it is better to use true weather as the training data. The possible reason is that the weather forecasting provides a coarse-grained predicted values, namely, 6th, 12th, 18th, 24th hours in future.

5.3.2. Impact of network depth

Fig. 13 presents the impact of network depth. As the network goes deeper (i.e. the number of residual units increases), the RMSE of the model first decreases and then increases, demonstrating that the deeper network often has a better result because it can capture not only near spatial dependence but also distant one. However, when the network is very deep (e.g. number of residual unit ≥ 14), training becomes more difficult and the possibility of overfitting becomes larger.

5.3.3. Impact of filter size and number

The receptive field of a convolution is determined by the size of the filter used. We here change the size of the filter from 2×2 to 5×5. Fig. 14(a) shows that the larger filter size has the lower RMSE, demonstrating larger receptive field has better ability to model spatial dependency. From Fig. 14(b), we can observe that more filters better result.

5.3.4. Impact of temporal closeness, period, trend

We here verify the impact of temporal *closeness*, *period*, *trend* components on TaxiBJ, as shown in Fig. 15. Fig. 15(a) shows the effect of temporal *closeness* where we fix $l_p = 1$ and $l_q = 1$ but change l_c. For example, $l_c = 0$ means that we does not employ the *closeness* component, resulting in a very bad result: 33.47 ± 0.34. We can observe that RMSE first decreases and then increases as the length of closeness increases, indicating that $l_c = 4$ has the best performance. Fig. 15(b) depicts the effect of *period* where we set l_c as 3 and l_q as 1 but change l_p. We can see that $l_p = 1$ has the best RMSE. The model without the *period* component (i.e. $l_p = 0$) is worse than the model with $l_p = 2, 3$, but better than the $l_p = 4$ model, meaning that short-range periods are always beneficial, and long-range periods may be hard to model or not helpful. Fig. 15(c) presents the effect of *trend* where l_c and l_p are fixed to 3 and 2, respectively. We change l_q from 0 to 3. The curve points that the $l_q = 1$ model outperforms others. Similar to *period*, it is better to employ the *trend* component, but long-range trend may be not easy to capture or useless.
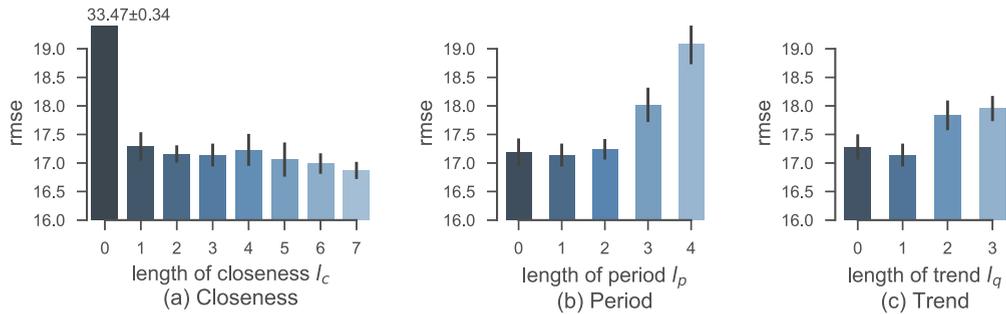
Fig. 15. Impact of temporal *closeness*, *period*, *trend*. We run each model 10 times where error bars show standard deviation.
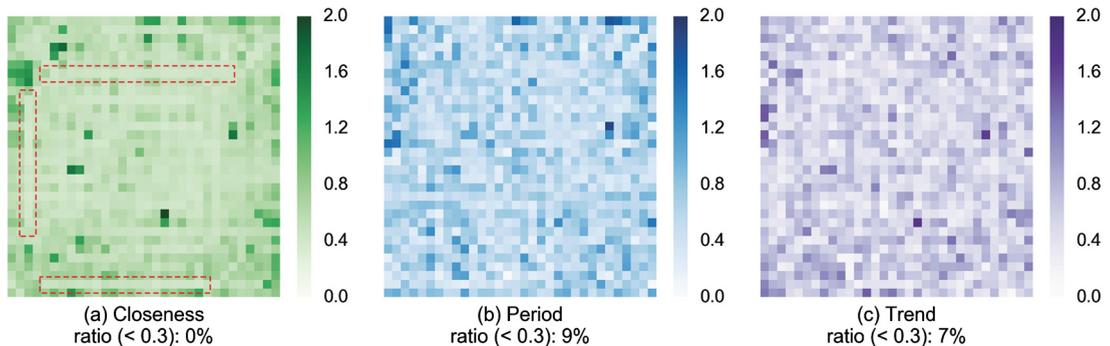
Fig. 16. Visualization of parameters.

To better understand the temporal *closeness*, *period* and *trend*, we here visualize the parameters of the parametric-matrix-based fusion layer, which is capable of learning different temporal influence degrees for each region of a city, as shown in Fig. 16. Each element in each sub-figure denotes a learned parameter of a certain region that reflects the influence degree by *closeness*, *period*, or *trend*. We here set a threshold (e.g. 0.3) to see the temporal properties of the whole city. Given a fixed threshold 0.3, we observe that the ratio (the number of regions whose parametric value is less than 0.3) of the *closeness* is 0, demonstrating all of regions in the city have a more or less *closeness*. The ratio of the *period* shows that there are 9% regions only have very weak periodic patterns. Likewise, Fig. 16(c) depicts that 7% regions do not have temporal *trend*. From Fig. 16(a), we find that the *closeness* of some main-road-related regions (red dashed frame) is not obvious. One reason is that the crowd flows in these regions can be predicted using *period* or/and *trend*, adding slight *closeness*.

5.3.5. Impact of activation function of output layer

In these experiments, we investigate the impact of activation functions (sigmoid and tanh) in the output layer. Since the range of the sigmoid function is $[0, 1]$, we have to scale the crowd flow data into $[0, 1]$. We run each model 5 times, and train for 50 epochs (without early stopping) using Adam [14]. Other hyperparameters are set as following: 12 (residual units), 32 (batch size), and 0.0002 (learning rate). Results are summarized in Fig. 17. We observe that ST-ResNet with tanh activation has faster convergence, yielding a better result (rmse: 16.80 ± 0.46) on the test set with fixed number of epochs.

Possible reason: the outputs of the sigmod functions are not zero-centered, and the corresponding inputs of the whole networks are also not zero-centered, which is undesirable since all layers[10] in the network would be receiving data that is not zero-centered. It has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive then the gradient on weights during back-propagation become either all positive, or all negative. Undesirable zig-zagging dynamics could happen in the gradient updates for the weights.

5.4. Evaluation of multi-step ahead prediction

According to Algorithm 2, we can use historical observations and the recent predicted ones to forecast the crowd flows in subsequent time intervals which is referred to multi-step ahead prediction. Fig. 18 shows multi-step prediction results of 12 different models on TaxiBJ. Among these models, ST-ResNet-BN and ST-ResNet-noExt are variants of ST-ResNet (12 residual units), where ST-ResNet-BN employs BN. LSTM-3, LSTM-6 and LSTM-12 are three variants of LSTM (see details in

---

[10] The outputs of middle layers with ReLU activation are non-zero-centered outputs.
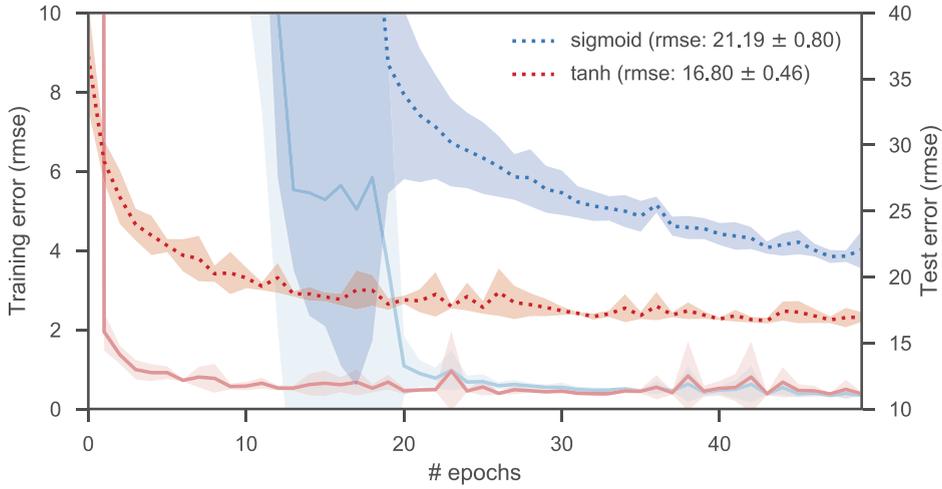
Fig. 17. Training curves on TaxiBJ of ST-ResNet. Dashed lines denote mean test error (y-axis on the right), solid lines denote mean training error (y-axis on the left), and shaded areas denote standard deviation. The Tanh makes ST-ResNet easier to train.
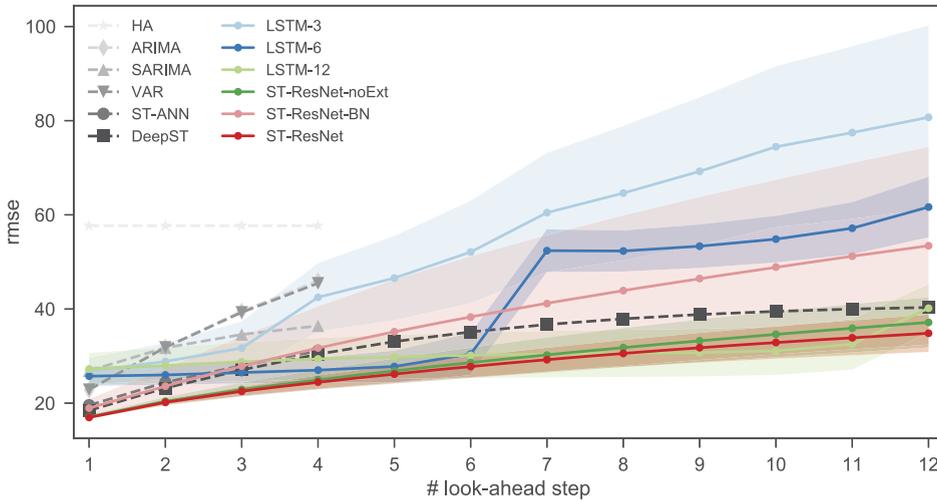
Fig. 18. Multi-step ahead prediction. For LSTM variants and ST-ResNet variants, we run each of them 10 times and use markers and shaded areas to denote mean RMSE and standard deviation, respectively.

Section 5.1). From the results of 4-step ahead prediction,[11] we find our ST-ResNet performs best though ST-ResNet-BN is better in the single-step ahead prediction, showing in Fig. 12(a). We observe that LSTM-12 is better than ST-ResNet when the number of the look-ahead steps is greater than 8. The reason may be that LSTM-12 uses the past 12 observations to predict, however, our ST-ResNet only takes past recent 3 observations as the input of the *closeness* component.

5.5. Efficiency and resources

We test the efficiency on two different virtual machines in the cloud (i.e. Microsoft Azure), as shown in Table 7. As introducing in Section 4.1, there are four main steps to predict crowd flows for each region of a city: (1) pulling trajectories from *Redis*; (2) converting trajectories into crowd flow data; (3) predicting the crowd flows in near future; (4) pushing results into *Redis*. We also report the time consumed by above four steps. Totally, A2 standard VM finishes the whole predicting process in 18.56 seconds. It takes 10.93 seconds on D4 standard VM, which is more powerful but expensive. One can choose different Cloud resources according to the requirements and budgets.

---

[11] 4-step ahead prediction on TaxiBJ means predicting the crowd flows in next 2 hours.

Table 7
Configuration of virtual machines and performance.

Virtual Machine (Azure)	A2 standard	D4 standard
OS	Ubuntu 14.04	Ubuntu 14.04
Memory	3.5 GB	28 GB
CPU	2 cores @ 2.20 GHz	8 cores @ 2.20 GHz
Keras version	1.1.1	1.1.1
Theano version	0.9.0dev	0.9.0dev
	Time (s)	
Pulling trajectories from redis	2.71	1.64
Converting trajectories into flows	9.65	6.05
Predicting the crowd flows	5.79	2.97
Pushing results into redis	0.41	0.27
Total	18.56	10.93

6. Related work

6.1. Crowd flow prediction

There are some previously published works on predicting an individual's movement based on their location history [21, 22]. They mainly forecast millions, even billions, of individuals' mobility traces rather than the aggregated crowd flows in a region. Such a task may require huge computational resources, and it is not always necessary for the application scenario of public safety. Some other researchers aim to predict travel speed and traffic volume on the road [23–25]. Most of them are predicting single or multiple road segments, rather than citywide ones [25,26]. Recently, researchers have started to focus on city-scale traffic flow prediction [20,27]. Both work are different from ours where the proposed methods naturally focus on the individual region not the city, and they do not partition the city using a grid-based method which needs a more complex method to find irregular regions first.

6.2. Classical models for time series prediction

Predicting the flows of crowds can be viewed as a type of time series prediction problem. There are several conventional linear models for such problem. The historical average model is portable, which simply uses the average value of historical time series to predict future value of time series. However, the model unable to respond to dynamic changes, such as incidents [28]. The Auto-Regressive Integrated Moving Average (ARIMA) model assumes that the future value of time series is a linear combination of previous values and residuals, furthermore, in order to obtain stationarity, the nonstationary time series should be differenced before analysis [29]. ARIMA is not suite for time series with missing data, since they relying on uninterrupted time series, and data filling technique might be problematic as the complexity of the situation increase [30]. The additional seasonal difference is often applied to seasonal time series to obtain stationarity before ARIMA being used, which is called SARIMA. The disadvantage of SARIMA is time consuming [30]. The Vector Autoregressive (VAR) models capture the linear inter dependencies among interrelated time series [31]. However, the correlation between predicted values and residuals is neglected.

Being different from the above linear models, the artificial neural network (ANN) model is a nonlinear model and commonly used in time series prediction [32–34]. ANNs have excellent nonlinear modeling ability, but not enough for linear modeling ability [35].

6.3. Deep neural networks

Neural networks and deep learning [4,36,37] have gained numerous success in the fields such as compute vision [10, 38], speech recognition [39,40], and natural language processing [41]. For example, convolutional neural networks won the ImageNet [42] competition since 2012, and help AlphaGo [43] beat Go human champion.[12] Recurrent neural networks (RNNs) have been used successfully for sequence learning tasks [44]. The incorporation of long short-term memory (LSTM) [16] or gated recurrent unit (GRU) [17] enables RNNs to learn long-term temporal dependency. However, both kinds of neural networks can only capture spatial or temporal dependencies. Recently, researchers combined above networks and proposed a convolutional LSTM network [45] that learns spatial and temporal dependencies simultaneously. Such a network cannot model very long-range temporal dependencies (e.g. period and trend), and training becomes more difficult as depth increases.

In our previous work [3], a general prediction model based on DNNs was proposed for spatio-temporal data. In this paper, to model a specific spatio-temporal prediction (i.e. citywide crowd flows) effectively, we mainly propose employing

---

[12] https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol.

the residual learning and a parametric-matrix-based fusion mechanism. A survey on data fusion methodologies can be found at [46].

6.4. Urban computing

Urban computing [2], has emerged as a new research area, which aims to tackle urban challenges (e.g. traffic congestion, energy consumption, and pollution) by using the data that has been generated in cities (e.g. geographical data, traffic flow, and human mobility). A branch of research also partitions a city into grids, and then studies the traffic flow in each region of the city, such as predicting urban air quality [47,48], detecting anomalous traffic patterns [49], inferring missing air quality [50], forecasting of spatio-temporal data [3]. Besides, some researchers started to research on deep learning methods for urban computing applications. For example, Song et al. proposed a recurrent-neural-network-based model to predict the person's future movement [51]. Chen et al. proposes a deep learning model to understand how human mobility will affect traffic accident risk [52]. Both work are very different from ours in terms of approach and problem setting. To the best of our knowledge, in the field of urban computing, end-to-end deep learning for forecasting citywide crowd flows has never been done.

7. Conclusion and future work

We propose a novel deep-learning-based model (called ST-ResNet) for forecasting the flow of crowds in each and every region of a city, based on historical spatio-temporal data, weather and events. Our proposed ST-ResNet is capable of learning all spatial (*nearby* and *distant*) and temporal (*closeness*, *period*, and *trend*) dependencies as well as external factors (e.g. weather, event). We evaluate our model on two types of crowd flows in Beijing and NYC, achieving performances which are significantly beyond 9 baseline methods, confirming that our model is better and more applicable to the crowd flow prediction. The code and datasets have been released at: https://www.microsoft.com/en-us/research/publication/deep-spatio-temporal-residual-networks-for-citywide-crowd-flows-prediction. We develop a Cloud-based system, called UrbanFlow, that can monitor the real-time crowd flows and provide the forecasting crowd flows in near future using our ST-ResNet.

In the future, we will consider other types of flows (e.g. metro card swiping data, taxi/truck/bus trajectory data, and phone signals data), and use all of them to generate more types of flow predictions, and *collectively* predict all of these flows with an appropriate fusion mechanism.

References

[1] J. Zhang, Y. Zheng, D. Qi, Deep spatio-temporal residual networks for citywide crowd flows prediction, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[2] Y. Zheng, L. Capra, O. Wolfson, H. Yang, Urban computing: concepts, methodologies, and applications, ACM Trans. Intell. Syst. Technol. 5 (3) (2014) 38.
[3] J. Zhang, Y. Zheng, D. Qi, R. Li, X. Yi, DNN-based prediction model for spatio-temporal data, in: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31–November 3, 2016, 2016, pp. 1–4, http://doi.acm.org/10.1145/2996913.2997016.
[4] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[5] Y.A. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 9–48.
[6] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[7] M. Mathieu, C. Couprie, Y. LeCun, Deep multi-scale video prediction beyond mean square error, arXiv preprint, arXiv:1511.05440.
[8] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.
[9] V. Jain, J.F. Murray, F. Roth, S. Turaga, V. Zhigulin, K.L. Briggman, M.N. Helmstaedter, W. Denk, H.S. Seung, Supervised learning of image restoration with convolutional networks, in: 2007 IEEE 11th International Conference on Computer Vision, IEEE, 2007, pp. 1–8.
[10] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[11] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015, 2015, pp. 448–456, http://jmlr.org/proceedings/papers/v37/ioffe15.html.
[12] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning, ICML-10, 2010, pp. 807–814.
[13] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[14] D. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980.
[15] Urban Flow website, http://urbanflow.sigkdd.com.cn/.
[16] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[17] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proccedings of the 2014 Conference on EMNLP, 2014, pp. 1724–1734, http://aclweb.org/anthology/D/D14/D14-1179.pdf.

[18] F. Chollet, Keras, https://github.com/fchollet/keras, 2015.
[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: large-scale machine learning on heterogeneous distributed systems, arXiv preprint, arXiv:1603.04467.
[20] M.X. Hoang, Y. Zheng, A.K. Singh, Forecasting citywide crowd flows based on big data, in: ACM SIGSPATIAL 2016, 2016, https://www.microsoft.com/en-us/research/publication/forecasting-citywide-crowd-flows-based-big-data/.
[21] Z. Fan, X. Song, R. Shibasaki, R. Adachi, Citymomentum: an online approach for crowd behavior prediction at a citywide level, in: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, 2015, pp. 559–569.
[22] X. Song, Q. Zhang, Y. Sekimoto, R. Shibasaki, Prediction of human emergency behavior and their mobility following large-scale disaster, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 5–14.
[23] A. Abadi, T. Rajabioun, P.A. Ioannou, Traffic flow prediction for road transportation networks with limited traffic data, IEEE Trans. Intell. Transp. Syst. 16 (2) (2015) 653–662.
[24] R. Silva, S.M. Kang, E.M. Airoldi, Predicting traffic volumes and estimating the effects of shocks in massive transportation systems, Proc. Natl. Acad. Sci. 112 (18) (2015) 5643–5648.
[25] Y. Xu, Q.-J. Kong, R. Klette, Y. Liu, Accurate and interpretable bayesian mars for traffic flow prediction, IEEE Trans. Intell. Transp. Syst. 15 (6) (2014) 2457–2469.
[26] P.-T. Chen, F. Chen, Z. Qian, Road traffic congestion monitoring in social media with hinge-loss Markov random fields, in: 2014 IEEE International Conference on Data Mining, IEEE, 2014, pp. 80–89.
[27] Y. Li, Y. Zheng, H. Zhang, L. Chen, Traffic prediction in a bike-sharing system, in: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2015, p. 33.
[28] B.L. Smith, M.J. Demetsky, Traffic flow forecasting: comparison of modeling approaches, J. Transp. Eng. 123 (4) (1997) 261–266.
[29] G.E. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: Forecasting and Control, John Wiley & Sons, 2015.
[30] B.L. Smith, B.M. Williams, R.K. Oswald, Comparison of parametric and nonparametric models for traffic flow forecasting, Transp. Res., Part C, Emerg. Technol. 10 (4) (2002) 303–321.
[31] S.R. Chandra, H. Al-Deek, Predictions of freeway traffic speeds and volumes using vector autoregressive models, J. Intell. Transp. Syst. 13 (2) (2009) 53–72.
[32] L. Florio, L. Mussone, Neural-network models for classification and forecasting of freeway traffic flow stability, Control Eng. Pract. 4 (2) (1996) 153–164.
[33] M.S. Dougherty, M.R. Cobbett, Short-term inter-urban traffic forecasts using neural networks, Int. J. Forecast. 13 (1) (1997) 21–31.
[34] G.P. Zhang, Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing 50 (2003) 159–175.
[35] G.P. Zhang, M. Qi, Neural network forecasting for seasonal and trend time series, Eur. J. Oper. Res. 160 (2) (2005) 501–514.
[36] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[37] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, an MIT Press book in preparation, Draft chapters available at http://www.deeplearningbook.org/.
[38] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2015, pp. 91–99.
[39] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 6645–6649.
[40] D. Yu, L. Deng, Automatic Speech Recognition, Springer, 2012.
[41] Q.V. Le, T. Mikolov, Distributed representations of sentences and documents, in: ICML, vol. 14, 2014, pp. 1188–1196.
[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252, https://doi.org/10.1007/s11263-015-0816-y.
[43] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[44] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[45] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, W.-c. Woo, Convolutional LSTM network: a machine learning approach for precipitation nowcasting, in: Advances in Neural Information Processing Systems, 2015, pp. 802–810.
[46] Y. Zheng, Methodologies for cross-domain data fusion: an overview, IEEE Trans. Big Data 1 (1) (2015) 16–34.
[47] Y. Zheng, F. Liu, H.-P. Hsieh, U-air: when urban air quality inference meets big data, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2013, pp. 1436–1444.
[48] Y. Zheng, X. Yi, M. Li, R. Li, Z. Shan, E. Chang, T. Li, Forecasting fine-grained air quality based on big data, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 2267–2276.
[49] L.X. Pang, S. Chawla, W. Liu, Y. Zheng, On detection of emerging anomalous traffic patterns using GPS data, Data Knowl. Eng. 87 (2013) 357–373.
[50] X. Yi, Y. Zheng, J. Zhang, T. Li, ST-MVL: filling missing values in geo-sensory time series data, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, 2016, pp. 2704–2710, http://www.ijcai.org/Abstract/16/384.
[51] X. Song, H. Kanasugi, R. Shibasaki, DeepTransport: prediction and simulation of human mobility and transportation mode at a citywide level, in: IJCAI, 2016.
[52] Q. Chen, X. Song, H. Yamada, R. Shibasaki, Learning deep representation from big and heterogeneous data for traffic accident inference, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.