UrbanFM: Inferring Fine-Grained Urban Flows

Yuxuan Liang\textsuperscript{1,2}, Kun Ouyang\textsuperscript{1}, Lin Jing\textsuperscript{2}, Sijie Ruan\textsuperscript{2,3}, Ye Liu\textsuperscript{1}
Junbo Zhang\textsuperscript{3}, David S. Rosenblum\textsuperscript{1}, Yu Zheng\textsuperscript{2,3}
\textsuperscript{1}School of Computing, National University of Singapore, Singapore
\textsuperscript{2}School of Computer Science and Technology, Xidian University, Xi’an, China
\textsuperscript{3}JD Intelligent City Research & JD Urban Computing Business Unit, Beijing, China
\{yuxliang,amber_jing,sijieruan,msjunbozhang,msyzheng\}@outlook.com,\{david,ouyangk,liuye\}@comp.nus.edu.sg

ABSTRACT

Urban flow monitoring systems play important roles in smart city efforts around the world. However, the ubiquitous deployment of monitoring devices, such as CCTVs, induces a long-lasting and enormous cost for maintenance and operation. This suggests the need for a technology that can reduce the number of deployed devices, while preventing the degeneration of data accuracy and granularity. In this paper, we aim to infer the real-time and fine-grained crowd flows throughout a city based on coarse-grained observations. This task is challenging due to the two essential reasons: the spatial correlations between coarse- and fine-grained urban flows, and the complexities of external impacts. To tackle these issues, we develop a method entitled UrbanFM based on deep neural networks. Our model consists of two major parts: 1) an inference network to generate fine-grained flow distributions from coarse-grained inputs by using a feature extraction module and a novel distributional upsampling module; 2) a general fusion subnet to further boost the performance by considering the influences of different external factors. Extensive experiments on two real-world datasets, namely TaxiBJ and HappyValley, validate the effectiveness and efficiency of our model compared to seven baselines, demonstrating the state-of-the-art performance of our approach on the fine-grained urban flow inference problem.

CCS CONCEPTS

•Computing methodologies → Neural networks; •Information systems applications → Spatial-temporal systems; •Applied computing → Transportation;

KEYWORDS

Urban computing, Deep learning, Spatio-temporal data, Urban flows

1 INTRODUCTION

The fine-grained urban flow monitoring system is a crucial component of the information infrastructure system in smart cities, which lays the foundation for urban planning and various applications such as traffic management. To obtain data at a spatial fine-granularity, the system requires large amounts of sensing devices to be deployed in order to cover a citywide landscape. For example, thousands of piezoelectric sensors and loop detectors are deployed on road segments in a city to monitor vehicle traffic flow volumes in real time; many CCTVs are deployed ubiquitously for surveillance purposes and for obtaining real-time crowd flow data. With a large number of devices deployed, a high cost would be incurred due to the long-term operation (e.g., electricity and communication cost) and maintenance (e.g., on-site maintenance and warranty). A recent study showed that in Anyang, Korea, the annual operation and device maintenance fee for their smart city projects reached 100K USD and 400K USD respectively in 2015 [15]. With the rapid development of smart cities on a worldwide scale, the cost of manpower and energy will become a prohibitive factor for the further intelligentization of the Earth. To reduce such expense, people require a novel technology which allows cutting the number of deployed sensors while, most importantly, keeping the original data granularity unchanged. Therefore, how to approximate the original fine-grained information from available coarse-grained data (obtained from fewer sensors) becomes an urgent problem.

Figure 1: Traffic flows in two granularities in Beijing, where each grid denotes a region. Our aim is to infer (b) given the coarse-grained data source (a).

Take monitoring traffic in a university campus as a regional example. We can reduce the number of the interior loop detectors and keep sensors only at the entrances of the campus to save cost. However, we still desire to recover the fine-grained flow distribution within the campus given only the coarse-grained information. In this paper, our goal is to infer the real-time and spatially fine-grained flow from observed coarse-grained data on a citywide scale with many other regions (as shown in Figure 1(a)). This Fine-grained Urban Flow Inference (FUFI) problem, however, is very challenging due to the reasons as follows:
Spatial Correlations. The fine-grained flow maps preserve spatial and structural correlations with the coarse-grained counterparts. Essentially, the flow volume in a coarse-grained superregion (e.g., the campus), is distributed to the constituent subregions (e.g., libraries) in the fine-grained situation. This implies a crucial structural constraint: the sum of the flow volumes in subregions strictly equals that of the corresponding superregion. Besides, the flow in a region can be affected by the flows in the nearby regions, which will impact the inference for the fine-grained flow distributions over subregions. Methods failing to capture these features would lead to a degenerated performance.

External Factors. The distribution of the flows in a given region is affected by various external factors, such as local weather, time of day and special events. To see the impact, we present a real-world study in an area of Beijing, as shown in Figure 2(a). On weekdays, (b) shows more flows occur in the office area and attractions at 10 a.m. as compared to at 8 p.m. where residences witness much higher flow density than the others (see (c)); on weekends, however, (c) demonstrates that people tend to present in a park area in the morning. It corresponds to our common sense that people go out for work in the morning, to attractions for relaxation in the weekend and return home at night. Besides, (d) shows that people keen to move to indoor areas instead of the outdoor park under storms. These observations evince that regions with different semantics present different flow distributions in respect of different external factors. Moreover, these external factors can intertwine and thus influence the actual distribution in a very complicated manner.

The FUFI problem can be cast as a mapping problem which maps data of low information entropy to that of high information entropy. Sharing the same nature with FUFI, recent studies [2, 14, 17] in image super-resolution (SR) have presented techniques for recovering high-resolution images from low-resolution images, which has motivated applications in other fields, such as meteorology [26]. Nevertheless, due to the aforementioned challenges, the simple application of these techniques to FUFI is infeasible and thus requires a careful redesign of the model architecture.

To this end, we present Urban Flow Magnifier (UrbanFM), a deep neural network model which learns to infer fine-grained urban flows under the supervised-learning paradigm. Following related techniques [2, 14, 17, 26], we assume a number of fine-grained data are available to bootstrap our solution. The key contributions of this paper lie in the following aspects:

- We present the first attempt to formalize the fine-grained urban flow inference problem with identification of the problem specificities and relevant challenges.
- We design an inference network to handle the spatial correlations. The inference network employs a convolutional network-based feature extraction module to address the nearby region influence. More importantly, it leverages a distributional upsampling module with a novel and parameter-free layer entitled $N^2$-Normalization to impose the structural constraint on the model, by converting the learning focus from directly generating flow volumes (as in related arts) to inferring the actual flow distribution.
- We design an external factor fusion subnet to account for all complex external influences at once. The subnet generates an integrated, high-level representation for external factors. The hidden features are then fed into different levels of the inference network (i.e., coarse- and fine-grained levels) to enhance the inference performance.
- We process, analyze, and experiment in two real-world urban scenarios, including the taxi flows with a metropolitan coverage and the human flows within a touristic district respectively. Our experimental results verify the significant advantages of UrbanFM over five state-of-the-art and two heuristical methods in both effectiveness and efficiency. Moreover, the experiments from multiple prospectives validate the rationale for different components of the model. We have released the code, sample data and demo for public use.

2 FORMULATION

In this section, we first define the notations and then formulate the problem of Fine-grained Urban Flow Inference (FUFI).

Definition 1 (Region) As shown in Figure 1, we partition an area of interest (e.g., a city) evenly into a $I \times J$ grid map based on the longitude and latitude, where a grid denotes a region [29]. Partitioning the city into smaller regions (i.e., using larger $L$, $J$) suggests that we can obtain flow data with more details, which results in a more fine-grained flow map.

Definition 2 (Flow Map) Let $X \in \mathbb{R}^{I \times J}$ represent a flow map of a particular time, where each entry $x_{i,j} \in \mathbb{R}_+$ denotes the flow volume of the instances (e.g., vehicle, people, etc.) in region $(i,j)$.

Definition 3 (Superregion & Subregion) In our FUFI problem, a coarse-grained grid map indicates the data granularity we can observe upon sensor reduction. It is obtained by integrating nearby grids within an $N$-by-$N$ range in a fine-grained grid map given a scaling factor $N$. Figure 1 illustrates an example when $N = 2$. Each coarse-grained grid in Figure 1(a) is composed of $2 \times 2$ smaller grids from Figure 1(b). We define the aggregated larger grid as superregion, and its constituent smaller regions as subregions. Note that with this setting, the superregions do not share subregions. Hence, the structure between superregions and the corresponding subregions indicates a special structural constraint in FUFI.

1The original data can be obtained through previously deployed sensors or from crowd-sourcing.

2https://github.com/yoshall/UrbanFM
3 METHODOLOGY

Figure 3 depicts the framework of UrbanFM, which consists of two main components for conducting the structurally constrained inference and capturing complex external influence, respectively. The inference network takes the coarse-grained flow map $X_c \in \mathbb{R}^{I \times J}$ as input, then extracts high-level features across the whole city by leveraging deep residual networks [7]. Taking extracted features as a priori knowledge, the distributional upsampling module outputs a flow distribution over subregions with respect to each superregion by introducing a dedicated $N^2$-Normalization layer. Finally, the Hadamard product of the inferred distribution with the upsamped coarse-grained flow map gives the fine-grained flow map $X_f$ as the network output. In the external factor fusion branch, we leverage embeddings and a dense network to extract pixel-wise external features in both coarse and fine granularity. The integration of external and flow features enables UrbanFM to exhibit fine-grained flow inference more effectively. In this section, we articulate the key designs for the two components, as well as the optimization scheme on network training.

3.1 Inference Network

Inference network aims to produce the fine-grained flow distribution over subregions from a coarse-grained input. We follow the general procedure in SR methods, which is composed of two phases: 1) feature extraction; 2) inference upon upsampling.

3.1.1 Feature Extraction. In the input stage, we use a convolutional layer (with $9 \times 9$ filter size, $F$ filters) to extract low-level features from the given coarse-grained flow map $X_c$, and perform the first stage fusion if external features are provided. Then $M$ Residual Blocks with identical layout take the (fused) low-level feature maps as input and then construct high-level feature maps. The residual block layout, as shown on the top right of Figure 3, follows the guideline in [14], which contains two convolutional layers ($3 \times 3$, $F$) followed by Batch Normalization [10], with an intermediate ReLU [6] function to introduce non-linearity.

Since we utilize a fully convolutional architecture, the reception field grows larger as we stack the network deeper. In other words, each pixel at the high-level feature map will be able to capture distant or even citywide dependencies. Moreover, we use another convolutional layer ($3 \times 3$, $F$) followed by batch normalization to guarantee feature extraction. Finally, drawing from the intuition that the output flow distribution would exhibit region-to-region dependencies to the original $X_c$, we employ a skip connection to introduce identity mapping [8] between the low-level features and high-level features, building an information highway skipping over the residual blocks to allow efficient gradient back-propagation.

3.1.2 Distributional Upsampling. In the second phase, the extracted features first go through $n$ sub-pixel blocks to perform an $N = 2^n$ upsampling operation which produces a hidden feature $H_f \in \mathbb{R}^{F \times N I \times NJ}$. The sub-pixel block, as illustrated in Figure 3, leverages a convolutional layer ($3 \times 3$, $F \times 2^2$) followed by batch normalization to extract features. Then it uses a PixelShuffle layer [19] to rearrange and upsample the feature maps to $2 \times$ size and applies a ReLU activation at the end. After each sub-pixel block, the output feature maps grow 2 times larger with the number of channels unchanged. A convolutional layer ($9 \times 9$, $F_0$) is applied post-upsampling, which maps $H_f$ to a tensor $H_{f_{\text{up}}} \in \mathbb{R}^{F_0 \times N I \times NJ}$. Note that $F_0 = 1$ in our case. In SR tasks, $H_{f_{\text{up}}}$ is usually the final output for the recovered image with super-resolution. However, the structural constraint essential to UFI hasn’t been considered.
Algorithm 1: $N^2$-Normalization

**Input:** x, scale_factor, $\epsilon$

**Output:** out

// x: an input feature map
// scale_factor: the upsampling factor
// $\epsilon$: a small number for numerical stability
// out: the structural distributions

sum = SumPooling(x, scale_factor);
sum = NearestNeighborUpsampling(sum, scale_factor);
out = x $\odot$ (sum+$\epsilon$) // element wise division

In order to impose the structural constraint on the network, one straightforward manner is to add a structural loss $L_s$ as a regularization term to the loss function:

$$L_s = \sum_{i,j} \left\| x_{i,j}^e - \sum_{i',j'} \alpha_{i',j'} x_{i,j}^f \right\|_F \quad \text{s.t.} \left\lfloor \frac{i'}{N} \right\rfloor = i, \left\lfloor \frac{j'}{N} \right\rfloor = j.$$  (2)

However, simply applying $L_s$ does not improve the model performance, as we will demonstrate in the experiments section. Instead, we design a $N^2$-Normalization layer, which outputs a distribution over every patch of $N$-by-$N$ subregions with regard to the respective superregion. To achieve this, we reformulate Equation 1 as follows:

$$x_{i,j}^f = \sum_{i',j'} \alpha_{i',j'} x_{i,j}^e$$  (3)

The volume flow in each subregion is now expressed as a fraction of that in the superregion, i.e., $x_{i',j'}^f = \alpha_{i',j'} x_{i,j}^e$, and we can treat the fraction as a probability. This allows us to interpret the network output in a meaningful way: the value in each pixel states how likely the overall flow will be allocated to the subregion $(i',j')$. By reformulation, we shift our focus from directly generating the fine-grained flow to generating the flow distribution. This essentially changes the network learning target and thus diverges from the traditional SR literature. To this end, we present the $N^2$-Normalization layer: $N^2$-Normalization($H^f_e$) = $H^f_e$, such that

$$H^f_{\pi(i,j)} = \sum_{r'\in[[N+1]/N-1]} H^f_{\alpha(i,j)^{r'}}$$  (4)

$N^2$-Normalization layer induces no extra parameters for the network. Moreover, it can be easily implemented within a few lines of code (see Algorithm 1). Also, the operations can be fully parallelized and automatically differentiated in runtime. Remarkably, this reformulation release the network from concerning varying output scales and enable it to focus on producing a probability within $[0,1]$ constraint.

Finally, we upscale $X^e$ using nearest-neighbor upsampling\(^3\) with the scaling factor $N$ to obtain $X^e_{up} \in \mathbb{R}^{N^2 \times N \times N}$ and then generate the fine-grained inference by $\tilde{X}^f = X^e_{up} \odot H^f_e$.

3.2 External Factor Fusion

External factors, such as weather, can have a complicated and vital influence on the flow distribution over the subregions. For instance, even if the total population in town remains stable over time, under storming weather people tend to move from outdoor regions to indoor regions. When different external factors entangle, the actual impact on the flow becomes implicit however unneglectable. Thereby, we design a subnet to handle those impacts \textit{all at once}.

Particularly, we first separate the available external factors into two groups, \textit{i.e.}, continuous and categorical features. Continuous features including temperature and wind speed are directly concatenated to be a vector $e_{con}$. As shown in Figure 3, categorical features include the day of week, the time of the day and weather (e.g., sunny, rainy). Inspired by previous studies [16], we transform each categorical attribute into a low-dimensional vector by feeding them into different embedding layers separately, and then concatenate those embeddings to construct the categorical vector $e_{cat}$. Then, the concatenation of $e_{con}$ and $e_{cat}$ gives the final embeddings for external factors, \textit{i.e.}, $e = [e_{con}; e_{cat}]$.

Once we get the concatenation vector $e$, we feed it into a feature extraction module, whose structure is depicted in Figure 3. By using dense layers, different external impacts are compounded to construct a hidden representation, which models the complicated interaction. The module provides two outputs: the coarse-grained feature maps $H^c_f$ and the fine-grained feature maps $H^f_f$, where $H^f_f$ is obtained by passing $X^e$ through $n$ sub-pixel blocks which are similar to the ones in the inference network. Intuitively, $H^c_f$ ($H^f_f$) is the spatial encoding for $e$ in coarse-grained (fine-grained) setting, modeling how each superregion (subregion) individually responds to the external changes. Therefore we concatenate $H^c_f$ with $X^e$, and $H^f_f$ with $\tilde{X}^f$ to the inference network. The early fusion of $H^c_f$ and $X^e$ allows the network to learn to extract a high-level feature describing not only the citywide flow, but also the external influences. Besides, the fine-grained $H^f_f$ carries the external information all the way to the rear of the inference network, playing a similar role as an information highway, which prevents information perishing in the deep network.

3.3 Optimization

UrbanFM provides an end-to-end mapping from coarse-grained input to fine-grained output, which is differentiable everywhere. Therefore, we can learn the network through auto back propagation, by providing training pairs ($X^c$, $\tilde{X}^f$) and calculating empirical loss between ($X^c$, $\tilde{X}^f$), where $\tilde{X}^f$ is the ground truth and $\tilde{X}^f$ is the outcome inferred by our network. Pixel-wise Mean Square Error (MSE) is a widely used cost function in many tasks, and we employ the same in this work as follows:

$$L(\Omega) = \|X^f - \tilde{X}^f\|_F^2$$  (5)

where $\Omega$ denotes the set of parameters in UrbanFM. Noted that $M$ and $F$ are the two main hyperparameters controlling the learning ability as well as the parameter size of the network. We experiment with different hyperparameter settings in the next section.

\(^3\)https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation
4 EXPERIMENTS

The focus of our experiments is on examining the capacity of our model in a citywide scenario. Therefore, we conduct extensive experiments using taxi flows in Beijing to comprehensively test the model from different aspects. In addition, we conduct further experiments in a theme park, namely Happy Valley, to show the model adaptivity on a relatively small area.

4.1 Experimental Settings

4.1.1 Datasets. Table 1 details the two datasets we use, namely TaxiBJ and HappyValley, where each dataset contains two sub-datasets: urban flows and external factors. Since a number of fine-grained flow data are available as ground truth, in this paper, we obtain the coarse-grained flows by aggregating subregion flows from the fine-grained counterparts.

- TaxiBJ: This dataset, which is published by Zhang et al. [29], indicates the taxi flows traveling throughout Beijing. As depicted in Figure 1, the studied area is split into 32x32 grids, where each grid reports the coarse-grained flow information every 30 minutes within four different periods: P1 to P4 (detailed in Table 1). Here, we utilize the coarse-grained taxi flows to infer fine-grained flows with 4x resolution (N = 4). In our experiment, we partition the data into non-overlapping training, validation and test data by a ratio of 2:1:1 respectively for each period. For example, in P1 (7/1/2013-10/31/2013), we use the first two-month data as the training set, the next month as the validation set, and the last month as the test set.

- HappyValley: We obtain this dataset by crawling from an open website[4] which provides hourly gridded crowd flow observations for a theme park named Beijing Happy Valley, with a total 1.25×10^5 m^2 area coverage, from 1/1/2018 to 10/31/2018. As shown in Figure 4, we partition this area with 25x50 uniform grids in coarse-grained setting, and target a fine granularity at 50x100 with an upscaling factor N = 2. Note that in this dataset, one special external factor is the ticket price, including day prices and night prices, which are obtained from the official account in WeChat. Regarding the smaller area, crowd flows exhibits large variance across samples given the 1-hour sampling rate. Thus, we use a ratio of 8:1:1 to split training, validation and test set to provide more training data.

4.1.2 Evaluation Metrics. We use three common metrics for urban flow data to evaluate the model performance from different facets. Specifically, Root Mean Square Error (RMSE) is defined as:

\[
RMSE = \sqrt{\frac{1}{z} \sum_{s=1}^{z} \| X_f^s - \hat{X}_f^s \|^2_f},
\]

where z is the total number of samples, X_f^s is s-th the inferred value and \( \hat{X}_f^s \) is corresponding ground truth. Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are defined as:

\[
MAE = \frac{1}{z} \sum_{s=1}^{z} | X_f^s - \hat{X}_f^s |_{l_1} \quad \text{and} \quad MAPE = \frac{1}{z} \sum_{s=1}^{z} \frac{|(X_f^s - \hat{X}_f^s) \odot X_f^s|}{|X_f^s|}_{l_1}.
\]

In general, RMSE favors spiky distributions, while MAE and MAPE focus more on the smoothness of the outcome. Smaller metric scores indicate better model performance.

### Table 1: Dataset Description.

| Dataset   | TaxiBJ | HappyValley |
|-----------|--------|-------------|
| Time span | P1: 7/1/2013-10/31/2013 | 1/1/2018 |
|           | P2: 2/1/2014-6/30/2014 | 1/1/2018 |
|           | P3: 3/1/2015-6/30/2015 | 10/31/2018 |
|           | P4: 11/1/2015-3/31/2016 | 10/31/2018 |
| Time interval | 30 minutes | 1 hour |
| Coarse-grained size | 32x32 | 25x50 |
| Fine-grained size | 128x128 | 50x100 |
| Upscaling factor (N) | 4 | 2 |

| External factors (meteorology, time and event) |
|-----------------------------------------------|
| Weather (e.g., Sunny) | 16 types | 8 types |
| Temperature/°C | [-24.6, 41.0] | [-15.0, 39.0] |
| Wind speed/mph | [0, 48.6] | [0.1, 15.5] |
| # Holidays | 41 | 33 |
| Ticket price/Y | / | [29.9, 260] |

### Figure 4: Visualization of crowd flows in HappyValley.

4.1.3 Baselines. We compare our proposed model with seven baselines that belong to the following three classes: (1) Heuristics. (2) Image super-resolution. (3) Meteorological super-resolution. The first two methods are designed by us based on intuition or empirical knowledge, while the next four methods are previously and currently state-of-the-art methods for single image super-resolution. The last method is the state of the art on statistical downscaling for climate data. We detail them as follows:

- **Mean Partition (Mean):** We evenly distribute the flow volume from each superregion in a coarse-grained flow map to the N^2 subregions, where N is the upsampling factor.

- **Historical Average (HA):** Similar to distributional upsampling, HA treats the value over each subregion a fraction of the value in the respective super region, where the faction is computed by averaging all training data.

- **SRCNN** [2]: SRCNN presents the first successful introduction of convolutional neural networks (CNNs) into the SR problems. It consists of three layers: patch extraction, non-linear mapping and reconstruction. Filters of spatial sizes 9x9, 5x5, and 5x5 were used respectively. The number of filters in the two convolutional layers are 64 and 32 respectively. In SRCNN, the low-resolution input is upscaled to the high-resolution space using a single filter (commonly bicubic interpolation) before reconstruction.

- **ESPCN** [19]: Bicubic interpolation used in SRCNN is a special case of the deconvolutional layer. To overcome the low efficiency of such deconvolutional layer, Efficient Sub-Pixel Convolutional Neural Network (ESPCN) employs a sub-pixel convolutional layer that aggregates the feature maps from LR space and builds the SR image in a single step.
Table 2: Results comparisons on TaxiBJ over different time spans (P1-P4).

| Methods        | P1       | P2       | P3       | P4       |
|----------------|----------|----------|----------|----------|
|                | RMSE     | MAE      | MAPE     | RMSE     | MAE      | MAPE     | RMSE     | MAE      | MAPE     |
| MEAN           | 20.918   | 12.019   | 4.469    | 20.918   | 12.019   | 5.364    | 27.442   | 16.029   | 5.612    | 19.049   | 11.070   | 4.192    |
| HA             | 4.741    | 2.251    | 0.336    | 5.381    | 2.551    | 0.334    | 5.594    | 2.674    | 0.328    | 4.125    | 2.023    | 0.323    |
| SRCNN          | 4.297    | 2.491    | 0.714    | 4.612    | 2.681    | 0.689    | 4.815    | 2.829    | 0.727    | 3.838    | 2.289    | 0.665    |
| ESPCN          | 4.206    | 2.497    | 0.732    | 4.569    | 2.727    | 0.732    | 4.744    | 2.862    | 0.773    | 3.728    | 2.228    | 0.711    |
| DeepSD         | 4.156    | 2.368    | 0.614    | 4.554    | 2.612    | 0.621    | 4.692    | 2.739    | 0.682    | 3.877    | 2.297    | 0.652    |
| VDSR           | 4.159    | 2.213    | 0.467    | 4.586    | 2.498    | 0.486    | 4.730    | 2.548    | 0.461    | 3.654    | 1.978    | 0.411    |
| SRResNet       | 4.164    | 2.457    | 0.713    | 4.524    | 2.660    | 0.688    | 4.690    | 2.775    | 0.717    | 3.667    | 2.189    | 0.637    |
| UrbanFM-ne     | 4.015    | 2.047    | 0.332    | 4.386    | 2.258    | 0.320    | 4.559    | 2.352    | 0.316    | 3.559    | 1.845    | 0.309    |
| UrbanFM        | 3.950    | 2.011    | 0.327    | 4.329    | 2.224    | 0.313    | 4.496    | 2.318    | 0.315    | 3.501    | 1.815    | 0.308    |

- **VDSR** [11]: Since both SRCNN and ESPCN follow a three-stage architecture, they have several drawbacks such as slow convergence speed and limited representation ability. Inspired by the VGG-net, Kim et al. presents a Super-Resolution method using Very Deep neural networks with depth up to 20. This study suggests that large depth is a necessary task for the SR.
- **SRResNet** [14]: SRResNet enhances VDSR by using the residual architecture presented by He et al. [7]. The residual architecture allows one to stack a much larger number of network layers, which bases many benchmark methods in computer vision tasks.
- **DeepSD** [26]: DeepSD is the state-of-the-art method on statistical downsampling for meteorological data. It basically exploits SRCNN for downsampling for an intermediate level, and performs further downsampling by simply stacking more SRCNNs. This method, however, would inherently require much more parameters compared with our method.

4.1.4 Variants. To evaluate each component of our method, we also compare it with different variants of UrbanFM:

- **UrbanFM-ne**: We simply remove the external factor fusion subnet from our method, which can help reveal the significance of this component.
- **UrbanFM-sl**: Upon removing the external subnet, we further replace distributional upsampling module by using sub-pixel blocks and $L_2$ to consider the structural constraint in this variant.

4.1.5 Training Details & Hyperparameters. Our model, as well as the baselines, are completely implemented by PyTorch with one TITAN V GPU. We leverage Adam [12], an algorithm for stochastic gradient descent, to perform network training with learning rate $\eta = 1e^{-4}$ and batch size being 16. We also apply a staircase-like schedule by halving the learning rate every 20 epochs, which allows smoother search near the convergence point. In the external subnet, there are 128 hidden units in the first dense layer with dropout rate 0.3, and $I \times J$ hidden units in the second dense layer. We embed DayOfWeek to $\mathbb{R}^2$, HourOfDay to $\mathbb{R}^3$ and weather condition to $\mathbb{R}^3$. Besides, for VDSR and SRResNet, we use the default settings in their paper. Since SRCNN, ESPCN and DeepSD perform poorly with default settings, we test different hyperparameters for them and finally use 768 and 384 as the number of filters in their two convolutional layers respectively. See more details in Appendix.

### 4.2 Results on TaxiBJ

#### Model Comparison

In this subsection, we compare the model effectiveness against the baselines. We report the result of UrbanFM with $M-F$ being 16-128 as our default setting. Further experiments regarding different $M-F$ will be discussed later. Likewise, we postpone the result of UrbanFM-sl to the next experiment for a more detailed study.

Table 2 summarizes the experimental results on TaxiBJ. We have the following observations: (1) The UrbanFM and its variant outperform all baseline methods in all three metrics over all time spans (P1-P4). Take SRResNet for example. UrbanFM advances it by 4.5%, 17.0% and 54.1% for RMSE, MAE and MAPE on average, where UrbanFM-ne also advances by 3.0%, 15.6% and 53.6% respectively. The advance of UrbanFM-ne over all baselines indicates that the combination with external subnet indeed enhances the model by incorporating external factors. (2) Image super-resolution methods outdo the heuristic method HA on RMSE while show deteriorate scores on MAE and MAPE. This can be attributed to two reasons: first, neural network methods are dedicated to performing well on RMSE as it is the training objective; second, HA preserves the spatial correlation for fine-grained flow maps while the others fail to do so. This again emphasizes the importance of preserving the structural constraint. A piece of further evidence can be seen from the comparison between UrbanFM-ne and SRResNet, where the former model has a similar structure as SRResNet except the distributional upsampling module, which makes it surpass its counterpart. Due to the similarity of model architecture, we select SRResNet as the baseline model for subsequent studies over different UrbanFM components.

Figure 5: Performance comparison over various structural constraints.
Study on Distributional Upsampling
To examine the effectiveness of the distributional upsampling module, we compared SRResNet with UrbanFM-ne (using distributional upsampling but no external factors) and UrbanFM-sl (using structural loss instead of distributional upsampling), as shown in Figure 5. In both M-F settings, it can be seen that UrbanFM-sl regularized by \( L_s \) performs very close to the SRResNet which is not constrained at all. Though under the setting of 16-64, Urban-sl achieves a smaller error than SRResNet in a subtle way, under the 16-128 setting they behave the opposite. On the contrary, UrbanFM-ne consistently outperforms the others on all three metrics. This result verifies that the distributional upsampling module is the better choice for imposing the structural constraint than using \( L_s \).

Study on External Factor Fusion
External impacts, though are complicated, can assist the network for better inferences when they are properly modeled and integrated, especially in a more difficult situation when there is less data budget. Thereby, we study the effectiveness of external factors under four difficulties by randomly subsampling different ratio of the original training set, which are 10%, 30%, 50% and 100%, corresponding to four levels: hard, semi-hard, medium and easy.

As shown in Figure 6, the gap between UrbanFM and UrbanFM-ne becomes larger as we reduce the number of training data, indicating that external factor fusion plays a more important role in providing a priori knowledge. When the training size grows, the weight for the priori knowledge decreases, as there exists overlaying information between observed traffic flow and external factors. Thus, the network may learn to capture some external impacts by providing enough data. Moreover, this trend also occurs between UrbanFM and UrbanFM-sl, which illustrates that the \( N^2 \)-Normalization layer provides a strong structural prior to facilitate the network training.

![Figure 6: Effects of external factors on four difficulties.](image)

Study on Parameter Size
Table 3 compares the average performance over P1 to P4. Across all hyperparameter settings, UrbanFM consistently outperforms SRResNet, advancing by at least 2.6%, 13.7% and 48.6%. Besides, this experiment reveals that adding more ResBlocks (larger \( M \)) or increasing the number of filters (larger \( F \)) can improve the model performance. However, these also increase the training time and memory space. Considering the tradeoff between training cost and performance, we set the default setting of UrbanFM to be 16-128.

Study on Efficiency
Figure 7 plots the RMSE on the validation set during the training phase using P1-100%. Figure 7(a) and 7(b) delineate that UrbanFM converges much smoother and faster than baselines and the variants. Specifically, 7(b) suggests such efficiency improvement can be mainly attributed to the \( N^2 \)-Normalization layer since UrbanFM-sl converges much slower and fluctuates drastically even it is constrained by \( L_s \), when compared with UrbanFM and UrbanFM-ne. This also suggests that learning the spatial correlation is a non-trivial task. Moreover, UrbanFM-ne behaves closely to UrbanFM as external factors fusion affects the training speed subtly when training data are abundant as suggested by the previous experiments.

![Figure 7: Convergence speed of various methods.](image)

Visualization
1) Inference error. Figure 8 displays the inference error \( \|X - \tilde{X}\|_1 \) from UrbanFM and the other three baselines for a sample, where a brighter pixel indicates a larger error. Contrast with the other methods, UrbanFM achieves higher fidelity for totality and in details. For instance, area A and B are "hard area" to be inferred, as A (Sanyuan bridge, the main entrance to downtown) and B (Shui bridge, a huge flyover) are two of the top congestion points in Beijing. Traffic flow of these locations usually fluctuates drastically and quickly, resulting in higher inference errors. Nonetheless, UrbanFM remains to produce better performance in these areas. Another observation is that the SR methods (SRCNN, ESPCN, VDSR and SRResNet) tend to generate blurry images as compared to structural methods (HA and UrbanFM). For instance, even if there is zero flow in area C, SR methods still generate error pixels as they overlap the predicted patches. This suggests the FUFI problem does differ from the ordinary SR problem and requires specific designs.

2) External influence. Figure 9(a)-(d) portray that the inferred distribution over subregions varies along with external factor changes. On weekdays, at 10 a.m., people had already flowed to the office area to start their work (b); at 9 p.m., many people had returned home after a hard-working day (c). On weekends, most people stayed home at 10 a.m. but some industrial researchers remained working in the university labs. This result proves that UrbanFM indeed captures the external influence and learns to adjust the inference accordingly.

| Methods     | Settings | #Params | RMSE | MAE  | MAFE |
|-------------|----------|---------|------|------|------|
| SRResNet   | 20-64    | 1.8M    | 4.317| 2.586| 0.725|
| UrbanFM    | 20-64    | 1.9M    | 4.094| 2.101| 0.321|
| SRResNet   | 16-64    | 1.5M    | 4.261| 2.520| 0.689|
| UrbanFM    | 16-64    | 1.7M    | 4.107| 2.118| 0.322|
| SRResNet   | 16-256   | 24.2M   | 4.178| 2.418| 0.614|
| UrbanFM    | 16-256   | 24.4M   | 4.068| 2.087| 0.316|
4.3 Results on HappyValley

Table 4 shows model performances using the HappyValley dataset. Note that in this experiment, we do not include DeepSD, since this task contains only 2x upscaling and the DeepSD degrades to SRCNN in this case. One important trait of the HappyValley dataset is that it contains more spikes on the fine-grained flow distribution, which results in a much larger RMSE score versus that in the TaxiBJ task. Nonetheless, UrbanFM remains the winner method outperforming the best baseline by 3.5%, 7.8% and 22%, and the UrbanFM-ne still holds the runner-up position. This proves that UrbanFM not only works on the large-scale scenario, but is also adaptable to smaller areas, which concludes our empirical studies.

Table 4: Results comparison on Happy Valley.

| Methods      | Settings | #Params | RMSE  | MAE   | MAPE  |
|--------------|----------|---------|-------|-------|-------|
| MEAN         | x        | x       | 9.206 | 2.269 | 0.799 |
| HA          | x        | x       | 8.379 | 1.811 | 0.549 |
| SRCNN       | 768      | 7.4M    | 8.291 | 2.175 | 0.816 |
| ESPCN       | 768      | 7.5M    | 8.156 | 2.155 | 0.805 |
| VDSR        | 20-64    | 0.6M    | 8.490 | 2.128 | 0.756 |
| SRResNet    | 16-128   | 5.5M    | 8.318 | 1.941 | 0.679 |
| UrbanFM-sl  | 16-128   | 5.5M    | 8.312 | 1.939 | 0.677 |
| UrbanFM-ne  | 16-128   | 5.5M    | 8.138 | 1.816 | 0.537 |
| UrbanFM     | 16-128   | 5.6M    | 8.030 | 1.790 | 0.531 |

5 RELATED WORK

5.1 Image Super-Resolution

Single image super-resolution (SISR), which aims to recover a high-resolution (HR) image from a single low-resolution (LR) image, has gained increasing research attention for decades. This task finds direct applications in many areas such as face recognition [5], fine-grained crowdsourcing [24] and HDTV [18]. Over years, many SISR algorithms have been developed in the computer vision community. To tackle the SR problem, early techniques focused on interpolation methods such as bicubic interpolation and Lanczos resampling [3]. Also, several studies utilized statistical image priors [22, 23] to achieve better performances. Advanced works aimed at learning the non-linear mapping between LR and HR images with neighbor embedding [1] and sparse coding [25, 28]. However, these approaches are still inadequate to reconstruct realistic and fine-grained textures of images.

Recently, a series of models based on deep learning have achieved great success in terms of SISR since they do not require any human-engineered features and show the state-of-the-art performance. Since Dong et al. [2] first proposed an end-to-end mapping method represented as CNNs between the low-resolution (LR) and high-resolution (HR) images, various CNN based architectures have been studied for SR. Among them, Shi et al. [19] introduced an efficient sub-pixel convolutional layer which is capable of recovering HR images with very little additional computational cost compared with the deconvolutional layer at training phase. Inspired by VGG-net for ImageNet classification [20], a very deep CNN was applied for SISR in [11]. However, training a very deep network for SR is really hard due to the small convergence rate. Kim et al. [11] showed residual learning speed up their training phase and verified that increasing the network depth could contribute to a significant improvement in SR accuracy.

Despite good performance on the RMSE accuracy, the generated image remains smooth and blurry. To address this problem, Ledig et al. [14] first proposed a perceptual loss function which consists of an adversarial loss to push their solution to the natural image manifold, and a content loss for the better reconstruction of high-frequency details. Lim et al. [17] developed an enhanced deep SR network that shows the state-of-the-art performance by removing unnecessary modules in [14]. Apart from super-resolving classical images, there are limited studies that focus on utilizing super-resolution methods to solve real-world problems in the urban area. For example, Vandal et al. [26], presented a stacked SRCNN [2] for statistical downscaling of climate and earth system simulations based on observational and topographical data.

However, these approaches are not suitable for the FUFI problem since the flow data present a very specific structural constraint with regard to natural images, as such, the related arts cannot be simply applied to our application in terms of efficiency and effectiveness. To the best of our knowledge, we are the first to formulate and solve the problem for fine-grained urban flow inference.
5.2 Urban Flows Analysis

Due to the wide applications of traffic analysis and the increasing demand for real-time public safety monitoring, urban flow analysis has recently attracted the attention of a large amount of researchers [31]. Over the past years, Zheng et al. [31] first transformed public traffic trajectories into other data formats, such as graphs and tensors, to which more data mining and machine learning techniques can be applied. Based on our observation, there were several previous works [4, 21] forecasting millions, or even billions of individual mobility traces rather than aggregated flows in a region.

Recently, researchers have started to focus on city-scale traffic flow prediction [9]. Inspired by deep learning techniques that power many applications in modern society [13], a novel deep neural network was developed by Zhang et al. [30] to simultaneously model spatial dependencies (both near and distant), and temporal dynamics of various scales (i.e., closeness, period and trend) for citywide crowd flow prediction. Following this work, Zhang et al. [29] further proposed a deep spatio-temporal residual network to collectively predict inflow and outflow of crowds in every city grid.

To address the data scarcity issue in crowd flows, very recent study [27] aims to transfer knowledge between different cities to help target city learn a better prediction model from the source city. Apart from the above applications, we aim to solve a novel problem (FUPI) on urban flows in this study.

6 CONCLUSION

In this paper, we have formalized the fine-grained urban flow inference problem and presented a deep neural network-based method (UrbanFM) to solve it. UrbanFM has addressed the two challenges that are specific to this problem, i.e., the spatial correlation as well as the complexities of external factors, by leveraging the original distributional upsampling module and the external factor fusion subnet. Experiments have shown that our approach advances baselines by at least 4.5%, 17.0% and 54.1% on TaxiBJ dataset and 3.5%, subnet. Experiments have shown that our approach advances baseline problem and presented a deep neural network-based method

In this paper, we have formalized the fine-grained urban flow inference problem and presented a deep neural network-based method (UrbanFM) to solve it. UrbanFM has addressed the two challenges that are specific to this problem, i.e., the spatial correlation as well as the complexities of external factors, by leveraging the original distributional upsampling module and the external factor fusion subnet. Experiments have shown that our approach advances baselines by at least 4.5%, 17.0% and 54.1% on TaxiBJ dataset and 3.5%, subnet. Experiments have shown that our approach advances baseline problem and presented a deep neural network-based method

REFERENCES

[1] Hong Chang, Dit-Yan Yeung, and Yinmin Xiong. 2004. Super-resolution through neighbor embedding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[2] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2016. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 38, 2 (2016), 295–307.

[3] Claude E Duchon. 1979. Lanczos filtering in one and two dimensions. Journal of Applied Meteorology 18, 8 (1979), 1016–1022.

[4] Zipei Fan, Xuan Song, Ryoitsuke Shibasaki, and Ryotaro Adachi. 2015. CityMo-mentum: an online approach for crowd behavior prediction at a citywide level. In Proceedings of the ACM International Joint Conference on Artificial Intelligence.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In European Conference on Computer Vision. 640–655.

[6] Minh X Hoang, Yu Zheng, and Ambuj K Singh. 2016. FCCF: forecasting city- wide flows based on big data. In Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 6.

[7] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015).

[8] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In The IEEE Conference on Computer Vision and Pattern Recognition. Vol. 2. 4.

[9] Sang Keon Lee, Heeso Rain Kwon, HeeAh Cho, Jongbok Kim, and Dongju Lee. 2016. International Case Studies of Smart Cities: Anyang, Republic of Korea (2016).

[10] Yuexuan Liang, Songyu Ke, Junbo Zhang, Xiwen Yi, and Yu Zheng. 2018. Geo-MAN: Multi-level Attention Networks for Geo-sensory Time Series Prediction. In Proceedings of the International Joint Conference on Artificial Intelligence.

[11] Bire Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In The IEEE Conference on Computer Vision and Pattern Recognition Workshops, Vol. 1. 4.

[12] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. 2003. Super-resolution image reconstruction: a technical overview. IEEE Signal Processing Magazine 20, 3 (2003), 21–36.

[13] Wenxue Shi, Jose Caballero, Ferenc Huszar, JOnsburgot, Andrew Cunningham, Alejandro Acosta, Andrew P Aitken, Alikhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Vol. 2. 4.

[14] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[15] Xuan Song, Quanhsi Zhang, Yoshihide Sekimoto, and Ryoitsuke Shibasaki. 2014. Prediction of human emergency behavior and their mobility following large-scale disaster. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 5–14.

[16] Jian Sun, Zongben Xu, and Heung-Yeung Shum. 2008. Image super-resolution using gradient profile prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 1874–1883.

[17] Yu-Wing Tai, Shuaicheng Liu, Michael S Brown, and Stephen Lin. 2010. Super-resolution using edge prior and single image detail synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2400–2407.

[18] Matt W Thornton, Peter M Atkinson, and DA Holland. 2006. Sub-pixel mapping of rural land cover objects from fine spatial resolution satellite sensor imagery using super-resolution pixel swapping. International Journal of Remote Sensing 27, 3 (2006), 473–491.

[19] Radu Timofte, Vincent De Smet, and Luc Van Gool. 2014. A+: Adjusted anchored neighborhood regression for fast super-resolution. In Asian Conference on Computer Vision. 111–126.

[20] Thomas Vandall, Evan Kodra, Sangram Ganguly, Andrew Michaelis, Ramakrishna Nemani, and Auropo R Ganguly. 2017. Deepgd: Generating high resolution climate change projections through single image super-resolution. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1663–1672.

[21] Leyi Wang, Xu Geng, Xiaojuan Ma, Feng Liu, and Qiang Yang. 2018. Crowd Flow Prediction by Deep Spatio-Temporal Transfer Learning. arXiv preprint arXiv:1802.00336 (2018).

[22] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. 2010. Image super-resolution using gradient profile prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 1–8.

[23] Yu-Wing Tai, Shuaicheng Liu, Michael S Brown, and Stephen Lin. 2010. Super-resolution using edge prior and single image detail synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2400–2407.

[24] Matt W Thornton, Peter M Atkinson, and DA Holland. 2006. Sub-pixel mapping of rural land cover objects from fine spatial resolution satellite sensor imagery using super-resolution pixel swapping. International Journal of Remote Sensing 27, 3 (2006), 473–491.

[25] Radu Timofte, Vincent De Smet, and Luc Van Gool. 2014. A+: Adjusted anchored neighborhood regression for fast super-resolution. In Asian Conference on Computer Vision. 111–126.

[26] Thomas Vandall, Evan Kodra, Sangram Ganguly, Andrew Michaelis, Ramakrishna Nemani, and Auropo R Ganguly. 2017. Deepgd: Generating high resolution climate change projections through single image super-resolution. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1663–1672.

[27] Leyi Wang, Xu Geng, Xiaojuan Ma, Feng Liu, and Qiang Yang. 2018. Crowd Flow Prediction by Deep Spatio-Temporal Transfer Learning. arXiv preprint arXiv:1802.00336 (2018).

[28] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. 2010. Image super-resolution using gradient profile prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 1–8.

[29] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In The AAAI Conference on Artificial Intelligence. 1655–1661.

[30] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In The AAAI Conference on Artificial Intelligence. 1655–1661.

[31] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In The AAAI Conference on Artificial Intelligence. 1655–1661.
A APPENDIX FOR REPRODUCIBILITY

To support the reproducibility of the results in this study, we have released our code and data\(^6\). Our implement is based on Pytorch 0.4.1. Here, we present the details of the dataset, normalization method and experimental settings.

A.1 Statistics of Datasets

In section 4, we have illustrated how we split the training, validation and test set based on the two real-world datasets: TaxiBJ and HappyValley. Since there are some coarse-grained data with most zero entries (i.e., extremely noisy data), we directly remove them from the original dataset. Here, we display the details of available samples in each set in Figure 5.

| Dataset   | Time Span | Size       |
|-----------|-----------|------------|
|           |           | train | valid | test       |
| TaxiBJ    | P1        | 1530  | 765   | 765        |
|           | P2        | 1797  | 891   |            |
|           | P3        | 1746  | 873   | 873        |
|           | P4        | 2122  | 1061  | 1061       |
| HappyValley| x         | 2188  | 273   | 275        |

A.2 Normalization Method

We employ data normalization before the training phase to speed up which is upsampled from the original coarse-grained \( f_\text{low} \) map. To support the reproducibility of the results in this study, we have released our code and data\(^6\). Our implement is based on Pytorch 0.4.1. Here, we present the details of the dataset, normalization method and experimental settings.

A.3 Detailed Settings of Baselines

We detail the model configuration as well as hyperparameter searching spaces for each baseline in this section.

- **Mean Partition (Mean):** It is parameter-free and can be directly applied on the test set.
- **Historical Average (HA):** Firstly, we compute the mean distribution matrix on the training set (with no parameters). Then, the matrix is applied to generate the fine-grained flow map over a coarse-grained observation.
- **SRCNN:** Since SRCNN under its default setting achieves inferior performance and takes a long time to converge, we test different hyperparameters for it, so as to find the best setting. Suppose there are \( F_1 \) and \( F_2 \) filters in the two convolutional layers of such method. We conduct a grid search over \( F_1 = \{64, 128, 256, 512, 768, 1024\} \) and \( F_2 = \{32, 64, 128, 256, 384, 512\} \). The setting in which \( F_1 = 768 \) and \( F_2 = 384 \) outperforms the others in the validation set.
- **ESPCN:** Similar to SRCNN with three-staged architecture, we leverage \( F_1 = 768 \) and \( F_2 = 384 \) as the number of filters in different convolutional layers respectively.
- **DeepSD:** Experiments show that ESPCN is more efficient and effective than SRCNN [19]. In order to speed up this method based on stacked SRCNNs, we use ESPCN to replace the SRCNN in the original paper, (i.e., a stacked ESPCN).
- **VDSR:** The depth of convolutional blocks \( D \) and the number of filters \( F \) in convolutional layer are two main hyperparameters. We utilize the default setting \( D = 20 \) and \( F = 64 \) as suggested by the authors [11].
- **SRResNet [14]:** In our paper, we compare our method with SRResNet from multiple angles. There are two main hyperparameters in SRResNet, including the depth of residual blocks \( M \) and number of filters in convolutional layer \( F \). We test \( M = \{16, 20\} \) and \( F = \{64, 128, 256\} \) in different experiments, which is detailed in Section 4.

A.4 Detailed Settings of UrbanFM

We first introduce how we implement \( N^2 \)-Normalization layer based on Pytorch, and further present the detailed settings of two main components of our approach, i.e., inference network and external factor fusion subnet.

A.4.1 \( N^2 \)-Normalization Layer

Figure 11 illustrates the Pytorch implementation of \( N^2 \)-Normalization layer, which plays a significant role in our method.

```python
1 def N2_Normalization(x, scale_factor, epsilon=1e-61):
2     # x: is the input feature map
3     # scale_factor: the upsampling factor
4     # epsilon: a small number to for numerical stability
5     # nn, torch: Pytorch packages
6     out = nn.AvgPool2d(scale_factor)(x) * scale_factor ** 2
7     out = nn.upsample(scale_factor=scale_factor, mode='nearest')(x)
8     out = torch.div(x, out + epsilon)
9     return out
```

Figure 11: Implementation of \( N^2 \)-Normalization layer based on PyTorch 0.4.1
A.4.2 Inference Network. Table 6 show the detailed configuration for the inference network which is depicted in Figure 3 (from left to right). Note that the upsampling factor $N = 4$ in this example.

Table 6: Details settings of Inference Network in Figure 3, where settings k-s-n means the size of kernel, stride and number of filters in a certain convolutional layer. We omit the batch size in the format of output for simplicity.

| Layer            | Setting | Output         |
|------------------|---------|----------------|
| Concat_1         | x       | $2 \times I \times J$ |
| Conv_1           | 9-1-64  | $64 \times I \times J$ |
| ReLU             | x       | $64 \times I \times J$ |
| Conv_1 (ResBlock)| 3-1-F   | $F \times I \times J$ |
| BatchNorm_1 (ResBlock) | x | $F \times I \times J$ |
| ReLU (ResBlock)  | x       | $F \times I \times J$ |
| Conv_2 (ResBlock)| 3-1-F   | $F \times I \times J$ |
| BatchNorm_2 (ResBlock) | x | $F \times I \times J$ |
| Other ResBlocks | ...     | ...            |
| Conv_2           | 3-1-F   | $F \times I \times J$ |
| BatchNorm        | x       | $F \times I \times J$ |
| Conv (SubPixel Block_1) | 3-1-4F | $4F \times I \times J$ |
| BatchNorm (SubPixel Block_1) | x | $4F \times I \times J$ |
| PixelShuffle (SubPixel Block_1) | x | $F \times 2I \times 2J$ |
| ReLU (SubPixel Block_1) | x | $F \times 2I \times 2J$ |
| Conv (SubPixel Block_2) | 3-1-4F | $4F \times 2I \times 2J$ |
| BatchNorm (SubPixel Block_2) | x | $4F \times 2I \times 2J$ |
| PixelShuffle (SubPixel Block_2) | x | $F \times 4I \times 4J$ |
| ReLU (SubPixel Block_2) | x | $F \times 4I \times 4J$ |
| Concat_2         | x       | $(F + 1) \times 4I \times 4J$ |
| Conv_3           | 9-1-1   | $1 \times 4I \times 4J$ |
| $N^2$-Normalization | x | $1 \times 4I \times 4J$ |

A.4.3 External Factor Fusion Subnet. Before inputting to the subnet, we use embedding method to convert the categorical features (like day of week, weather condition\(^7\)) to learned representations respectively, i.e., real-valued vectors. As shown in Table 7, we detail the embedding settings for each external factor.

Table 7: Embedding setting of external factors.

| Data         | Feature     | #Categories | Embed Length |
|--------------|-------------|-------------|--------------|
| Meteorology  | Temperature | x           | 1            |
| Wind speed   | x           | 1           |
| Weather      | 16 (8)      | 3           |
| Time         | Holiday     | 2           | 1            |
|              | Weekend     | 2           | 1            |
|              | Day of week | 7           | 2            |
|              | Hour of day | 24          | 3            |
| Event        | Ticket price| x           | 1            |

\(^7\)TaxiBJ witnesses 16 kinds of weather conditions: Sunny, Cloudy, Overcast, Rainy, Sprinkle, ModerateRain, HeavyRain, Rainstorm, Thunderstorm, FreezingRain, Snowy, LightSnow, ModerateSnow, HeavySnow, Foggy, Sandstorm, Dusty. For HappyValley, only 8 types of above weather conditions are included.

Table 8 shows the details of the external subnet. The settings of dense layer denotes the number of hidden units, while that of dropout layer represents its randomly dropping rate. It is also illustrated from left to right of Figure 3.

Table 8: Details of External Factor Fusion in Figure 3.

| Layer            | Setting | Output |
|------------------|---------|--------|
| Dense_1          | 128     | 128    |
| Dropout          | 0.3     | 128    |
| ReLU_1           | x       | 128    |
| Dense_2          | $I \times J$ | $I \times J$ |
| ReLU_2           | x       | $I \times J$ |
| Conv (SubPixel Block_1) | 3-1-4 | $4 \times I \times J$ |
| BatchNorm (SubPixel Block_1) | x | $4 \times I \times J$ |
| PixelShuffle (SubPixel Block_1) | x | $1 \times 2I \times 2J$ |
| ReLU (SubPixel Block_1) | x | $1 \times 2I \times 2J$ |
| Conv (SubPixel Block_2) | 3-1-4 | $4 \times 2I \times 2J$ |
| BatchNorm (SubPixel Block_2) | x | $4 \times 2I \times 2J$ |
| PixelShuffle (SubPixel Block_2) | x | $1 \times 4I \times 4J$ |
| ReLU (SubPixel Block_2) | x | $1 \times 4I \times 4J$ |