SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

November 2026, Vol. 69, Iss. 11, 212302:1–212302:19
https://doi.org/10.1007/s11432-025-4970-3

Can a CNN perform well on network traffic matrices? A case study on formulating network data to maximize the benefits of deep learning

Ruotian XIE[1,2], Kun XIE[1*], Xin WANG[3], Jigang WEN[4], Gaogang XIE[5] & Guangxing ZHANG[6]

[1]*College of Computer Science and Electronics Engineering, Hunan University, Changsha 410082, China*
[2]*School of Software, Nanchang University, Nanchang 330047, China*
[3]*Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook 11794, USA*
[4]*School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China*
[5]*Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China*
[6]*Department of Network Technology Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*

Received 10 September 2025/Revised 14 January 2026/Accepted 15 May 2026/Published online 20 October 2026

Abstract With deep learning (DL) revolutionizing networking research, it has become trendy to cast network problems (e.g., networking prediction and control) as machine-learning tasks. The prevailing belief holds that by inputting all relevant information into a DL model and training it with a fitting loss function, DL's remarkable capability will uncover an optimal solution. However, this optimistic view often does not hold true. This work presents a case study highlighting the necessity and challenges of carefully formulating network data to enable DL models to effectively understand network problems and capture domain-specific structures within a network. Traffic matrices (TMs), which detail the traffic volume for source-destination pairs and their temporal patterns, are crucial for network management tasks like traffic engineering and resource provisioning. Given their matrix structure and inherent spatial patterns, TMs might superficially resemble images, suggesting computer vision techniques like convolutional neural networks (CNNs) could be suitable for TM modeling and prediction. Yet, our attempts to train CNNs for TM interpolation were unsuccessful. We noted that CNNs excel at extracting features from data with spatial locality, which TMs lack due to the non-adjacent nature of correlated traffic patterns. To address this issue, we introduce a novel module that reorganizes TMs through weighted permutation matrices, thus enhancing the representation of traffic structure in a form more amenable to CNN processing. Our extensive testing confirms the effectiveness of our method in a range of TM-related tasks, including temporal interpolation, prediction, and completion.

Keywords traffic matrix, machine learning, AI for network, network measurement, network traffic

Citation Xie R T, Xie K, Wang X, et al. Can a CNN perform well on network traffic matrices? A case study on formulating network data to maximize the benefits of deep learning. Sci China Inf Sci, 2026, 69(11): 212302, https://doi.org/10.1007/s11432-025-4970-3

1 Introduction

1.1 Motivation and problem

The rapid advancement of deep learning (DL) techniques has garnered increasing interests in applying them to network studies [1–6]. In these studies, network problems are typically cast as machine learning tasks, incorporating a wide array of relevant data into the input of a DL model. A selected machine model is often trained with a carefully chosen loss function and leverages the remarkable pattern-fitting prowess of DL to approximate the optimal solution. However, without explicitly considering network-specific data structures and operational characteristics, many DL-based network models rely on increasingly complex architectures that are difficult to tune and often fail to generalize well in practice.

In networks, traffic matrices (TMs) [7] are a fundamental data abstraction that quantifies traffic volumes between source-destination pairs. TMs are routinely used by network operators as critical inputs for real-world tasks such as traffic engineering and capacity planning [8,9], traffic prediction [10,11], anomaly detection [12], network slicing [13], and path steering [14], directly influencing routing decisions, congestion mitigation, and service-level guarantees.

* Corresponding author (email: xiekun@hnu.edu.cn)

© Science China Press 2026

While graph neural networks (GNNs) are powerful for learning spatial dependencies, their effectiveness heavily relies on the availability of accurate graph topology information, such as link connectivity, routing weights, or node relationships. In many real-world networking scenarios, however, such information is either unavailable, dynamically changing, or expensive to collect, which significantly limits the practicality of GNN-based solutions.

In contrast, convolutional neural networks (CNNs) require no explicit topology or routing information and operate directly on matrix-form inputs, making them one of the most widely adopted and deployable architectures in networking applications. Given that TMs are naturally represented as matrices, several studies [10, 11, 13, 15, 16] have explored the use of CNNs for TM-related tasks. However, despite their simplicity and practicality, the direct application of CNNs to TMs often yields limited performance gains, failing to reproduce the success observed in image-related tasks.

Despite their shared matrix structure, TMs and images possess fundamentally different domain-specific structures. For instance, images exhibit spatial locality due to the color similarity of adjacent pixels. TMs, however, represent traffic flows between nodes identified by IDs rather than spatial coordinates, lacking inherent spatial locality and exhibiting traffic patterns that do not necessarily align with physical proximity.

Our research diverges from conventional TM-focused studies. We do not concentrate on specific applications or scenarios. Instead, we propose a novel ML framework that CNN can perform more effectively in accomplishing TM-related tasks: **reformatting TMs to enable CNNs to understand the domain-specific structures inherent in TMs, thereby unlocking the full potential of CNNs for TM analyses.**

1.2 Investigation summary & contribution

We use TMs and CNNs as a case study to demonstrate the importance of DL formulation in maximizing the benefits of DL in the network field. Given the significant roles TMs play in networking, enabling CNNs to effectively understand and extract features from TMs could greatly benefit the domain.

We have conducted a series of studies to answer the following three fundamental questions.

• **RQ1:** Can CNNs perform well when applied to learn from other matrix data (e.g., TMs) besides images?

• **RQ2:** If not, how can we rearrange the matrix data so that CNNs can better understand their specific structures and improve performance?

• **RQ3:** Can CNNs with rearranged matrices perform similarly or better than GNNs and Transformer-based models?

To evaluate how CNNs perform on other matrix data, we use network TMs as a case study and design a traffic temporal interpolation model (TIM). We run TIM on two network traffic datasets and, for comparison, a video dataset to evaluate the CNN-based TIM model. The experimental results (Table 1) reveal that, compared to video data, the interpolation accuracy on TMs is significantly worse. This indicates that CNNs work well on images but not on network TMs.

Both images and TMs contain matrix data. To understand why CNNs behave differently on these data types, we conduct additional experiments. After shuffling the rows and columns of images in the video, we test the CNN-based TIM model again. However, compared with the original video data, the interpolation accuracy significantly decreases under the shuffled video. Thus, we hypothesize that spatial locality in the matrix directly affects CNN performance, and CNNs cannot perform well on network TMs because spatial locality in TMs is weak.

To enhance the locality of TMs, we design an ant colony algorithm to test the performance of rearranging TMs under five popular similarity metrics. Unfortunately, although we find a metric suitable for video rearrangement, we cannot identify any metrics that work for TMs. Our results indicate that, despite matrix rearrangement being able to impact CNN performance, using similarity metrics to rearrange TMs is not a feasible solution for CNN enhancement. Furthermore, the underlying structures and characteristics of TMs and images are completely different; thus the finding of rearrangement solution suitable for all types of matrix data is a big challenge.

To discover a suitable rearrangement to make TMs compatible with CNNs and facilitate their understanding by CNNs, we analyze matrix multiplication. We find that rearranging the rows and columns of a matrix corresponds to left-multiplying the matrix by a row permutation matrix and right-multiplying it by a column permutation matrix. Based on this, we design a novel rearrangement and reinforcement (R&E) neural network framework to find the permutation matrices. Experimental results demonstrate that the proposed R&E network is very effective in increasing the locality of TMs, thus achieving highly appealing performance.

To explain why R&E can improve the locality of TMs, we visualize the permutation process under R&E (Figure A1). The visualization shows that our R&E can not only rearrange a matrix to enhance locality but also reinforce the locality through weighted matrix multiplications.

To determine if the R&E network is a general framework to augment CNN's ability to understand domain-specific structures for other tasks and matrix data, we apply it to two other CNN-based models for prediction and missing completion tasks. We also apply the R&E network to two urban sensing matrices (USMs) in the urban sensing field. As expected, all tasks perform much better with the help of our R&E network, demonstrating that the R&E network is promising for various tasks with different types of matrix data.

To verify that our R&E network can enhance CNN's ability to understand data features when CNNs are components in complex neural networks with other DL structures stacked, we integrate our R&E network into five existing CNN-based models for prediction. Experimental results (Appendix B, Table A1) show that our R&E network effectively enhances the accuracy of existing CNN-based methods from other studies while only adding a small burden to the model training process.

To validate whether CNN-based model with a rearranged matrix outperforms GNN-based and Transformer-based models, we compared the five CNN-based models with three GNN-based and two Transformer-based models (Appendix B, Table A1). The results are as follows. (1) CNN models, with the rearranged matrix, perform similarly or better than GNN-based models, even though without requiring graph information. (2) They also achieve comparable or superior performance to Transformer-based models.

Specifically, our proposed R&E network has the following key features.

• R&E network is a general technique to enhance CNN's understanding of domain-specific structures in different kinds of matrix data. Besides TM-based tasks, the R&E network is also helpful when used with TaxiBj data (i.e., USM data) and even for image data.

• The permutation matrices found in the R&E network have strong portability. Experiments show that the permutation matrices found in one task (i.e., temporal interpolation) can be directly used in other tasks (i.e., prediction, missing completion).

• R&E network has strong interpretability, as it is designed based on matrix multiplication. With the permutation matrices found by the R&E network, it is easy to determine which rows/columns are important in CNN's feature extraction and how the matrix's locality can be strengthened under our R&E network (Appendix A).

• Without requiring graph structural information, CNN with R&E can achieve similar or even better performance compared with GNN-based models, while also outperforming Transformer-based models.

2 Related work

This paper presents a case study that highlights the necessity and challenges of carefully formulating network data to enable DL models to effectively understand network problems and capture domain-specific structures.

Since both TMs and images are represented as matrices, CNNs have recently been applied to extract features from TMs [10,11,13,15,16]. For example, Ref. [15] proposed a CNN-based NTC method for estimating missing entries in TMs. Similarly, Ref. [16] developed the GS-STN model for traffic prediction, where CNN is employed to extract TM features that are subsequently fed into an LSTM to learn attention factors and predict future mobile traffic demands. Beyond these, CNNs have been combined with LSTMs and attention mechanisms for traffic prediction [10,11] and applied to traffic engineering [8]. These studies highlight CNNs' ability to extract matrix-based data features when integrated with other deep learning architectures.

Unlike prior studies, this paper addresses a novel problem: reformulating TMs to help CNNs better understand TM data, thereby enabling more effective feature extraction and improved learning performance across various tasks.

Besides CNNs, other DL techniques such as GNNs and Transformers have gained popularity. GNN-based methods [17–22] explicitly leverage graph structures and can achieve strong performance when accurate topology and auxiliary information are available. However, such information is often incomplete, dynamically changing, or costly to obtain in operational networks, which constrains the applicability of GNNs in practice. In contrast, CNNs require minimal structural assumptions and remain one of the most widely adopted architectures in deployed networking systems. Similarly, while Transformers [23,24] have become increasingly popular for various tasks, they typically require large training datasets and may not always be effective in specific scenarios [25,26], where CNNs still outperform [25]. Consequently, CNNs continue to serve as a reliable and foundational solution for spatial feature extraction.

Rather than proposing a new DL model to replace CNNs or designing task-specific architectures, this paper focuses on a complementary but less explored direction: reformulating TMs so that their latent domain-specific structures become amenable to convolutional operations, thereby enabling CNNs to better exploit their representation capacity across diverse TM-related tasks.

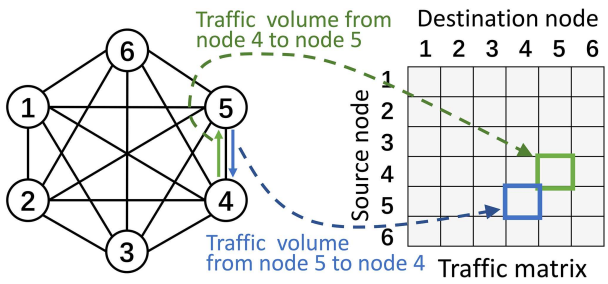

Figure 1 (Color online) Traffic matrix.

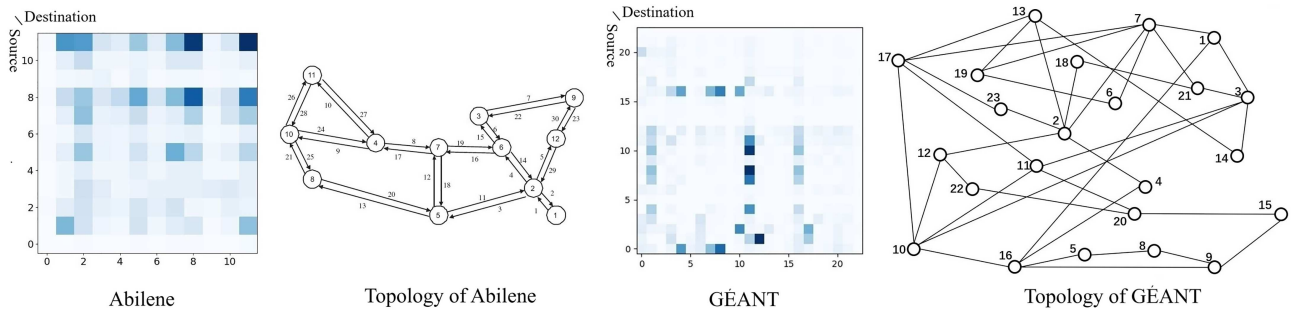

Figure 2 (Color online) Traffic matrices in Abilene, GÉANT.

Reordering or restructuring matrices has been explored in related communities [27–29], but existing efforts are predominantly task-driven and rely on predefined or domain-specific reordering rules. For example, ADnEV [28] performs matrix adjustment explicitly for schema matching, cross-association methods [27] target structure discovery and compression, and SDBN [29] reorders brain networks to support disease diagnosis. In all these cases, matrix reordering is designed to serve a specific downstream objective and is guided by fixed heuristics (e.g., similarity thresholds, clustering criteria, or spectral properties). As a result, a reordering strategy that is effective for one task or domain may not generalize to other tasks or data settings.

In contrast, our R&E network addresses a more general and previously unexplored problem in networking: how to automatically reformulate TMs so that CNNs can effectively capture their domain-specific structures across diverse TM-related tasks. Instead of relying on handcrafted rules or task-dependent objectives, we treat matrix reordering as a fully data-driven and self-adaptive process, where the optimal row and column permutation is learned end-to-end to maximize CNN feature extraction. This design enables the learned reordering to adapt to different tasks and scenarios, providing a general data formulation mechanism rather than a task-specific preprocessing step.

3 System model and problem

For a network with N nodes, an $N \times N$ TM X can be constructed to represent the traffic distribution with the rows and columns denoting the source nodes and destination nodes, respectively. The ij-th element x_{ij} denotes the amount of traffic flowed from node i to j. A node can be an individual router, or a site that contains multiple routers.

Figure 1 shows a 6×6 TM that records the volumes of traffic flows in the network of 6 nodes. For example, the element x_{45} (green box) represents the traffic volume flowing from node 4 to 5, the element x_{54} (blue box) represents the traffic volume flowing from node 5 to 4.

A TM is a traffic volume snapshot of the network at a specific time slot. Figure 2 shows two TMs of Abilene [30] and GÉANT [31]. Abilene is a high-performance backbone network created by the Internet2 community. The Abilene dataset consists of a sequence of 12×12 matrices, and records the traffic every 5 min for 7 months. GÉANT is the pan-European data network for the research and education community. The GÉANT data set is composed of a sequence of 23×23 matrices and records the traffic every 15 min for 112 days. In Figure 2, the darker the color of the element in the matrix, the larger the value is.

In the image domain, a digital grayscale image is represented by a pixel matrix (i.e., image matrix). Each image processing operation is an operation on the image matrix. CNNs are known for their effectiveness in image feature extraction and associated tasks.

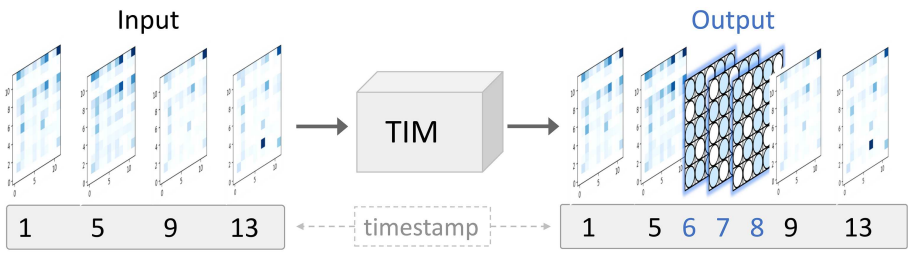

Figure 3 (Color online) Traffic temporal interpolation problem.

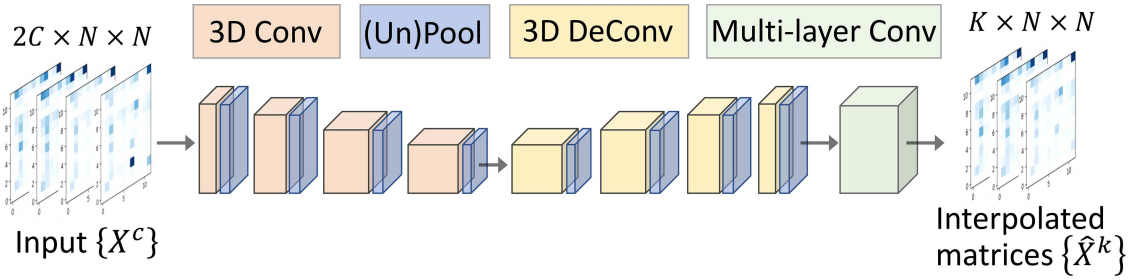

Figure 4 (Color online) Traffic temporal interpolation model (TIM) based on CNN.

As a TM shares some similarities with an image matrix, in this paper, we would like to achieve the following two goals. (1) Investigating the techniques that enable CNNs to understand TMs and be aware of their specific structures. (2) Ensuring CNN to work effectively over TMs, and answering the three fundamental questions mentioned in the introduction.

4 Experimental studies on traffic matrix interpolation

In this section, we take the interpolation task as a case study to compare the CNN performance between TM and image data, as outlined in RQ1. Our focus here lies in understanding CNN behavior rather than presenting a new interpolation model.

A TM X^k captures a traffic volume snapshot of a network at time slot k. By combining TMs of multiple consecutive time slots, we obtain a sequence of TMs. Due to the high cost of performing fine-grained traffic measurements in a network, the network monitoring system typically records a sequence of coarse-grained TMs, meaning the interval between two adjacent TMs in the sequence is large. For improved microburst detection and anomaly detection [12], a fine-grained traffic sequence with smaller intervals between TMs is required.

To generate a fine-grained traffic sequence from a coarse-grained traffic sequence and to investigate whether CNNs can effectively extract the features hidden in TMs, we first present our design of a traffic TIM based on CNN. We then evaluate the performance of TIM on two network traffic datasets, Abilene [30] and GÉANT [31], as well as one video dataset, TAA [32].

4.1 Traffic temporal interpolation model

Traffic temporal interpolation is a task that, given a coarse-grained traffic sequence, generates a fine-grained traffic sequence by calculating the intermediate TMs between consecutive TMs in the coarse-grained traffic sequence. Figure 3 serves as an illustration of the traffic temporal interpolation problem. It displays a coarse-grained traffic sequence composed of 4 TMs taken at time slots 1, 5, 9, and 13, which are input to the TIM. After performing traffic temporal interpolation, three new TMs at timestamps 6, 7, and 8 (Figure 3, blue font) are inferred, resulting in a fine-grained traffic sequence.

A video consists of a sequence of images (i.e., frames). To increase the frame rate, many methods on video frame interpolation have been proposed [33–38], most of which are designed based on CNN. Accordingly, we also design our traffic TIM based on CNN, as shown in Figure 4.

The TIM model utilizes a combination of 3D convolutions (orange blocks), deconvolutions (yellow blocks), and multi-layer convolutions to perform its operations. The 3D convolutions and deconvolutions are responsible for extracting features, while the multi-layer convolution (green block) is used to infer the interpolated TMs.

Table 1 Performance of TIM. The best results are in bold.

	AER	ANMAE	ARMSE
Abilene	0.29913	0.29001	0.00204
GÉANT	0.22454	0.2065	13028.1
TAA	**0.19522**	**0.13166**	12.1067

Let $\{X^c\}$ denote the input to the TIM model, which is a time sequence comprising $2C$ TMs. Each TM has a size of $N \times N$, where N represents the number of nodes in the network. The frames X^C and X^{C+1} correspond to the TMs located in the middle of the sequence, specifically at positions C and $C+1$, respectively. The TIM model is capable of generating K interpolated matrices $\hat{X}^k$ in the middle between X^C and X^{C+1} (refer to [38]). In Figure 3, we observe that $2C$ is equal to 4 and K is equal to 3. The left side of the TIM diagram depicts the original TM sequence with 4 TMs (i.e., $2C = 4$), while the right side of the TIM diagram illustrates the insertion of 3 additional TMs between X^C (TM 5) and X^{C+1} (TM 9). This results in a total of 7 TMs.

In the rest of our experiments, we set $2C$ to 8 and K to 1, which indicates that we input 8 TMs into the TIM model and generate 1 interpolated TM positioned in the middle of the TM sequence. To handle long sequences of TMs with a large number of TMs, we utilize a sliding window strategy in the TIM model. This strategy allows us to process the TMs in smaller batches and concatenate the interpolated results to obtain longer interpolated TMs. We train the TIM model using the MAE loss

$$MAE(\{X^k\}, \{\hat{X}^k\}) = \mathbb{E}_k \left[\left| X^k - \hat{X}^k \right| \right], \tag{1}$$

where $\{\hat{X}^k\}$ and $\{X^k\}$ are the sets of K interpolated TMs and K ground-truth TMs, respectively. $\mathbb{E}_k$ denotes expectation, $\hat{X}^k$ is the interpolated TM at position k, and X^k is the corresponding ground-truth.

4.2 Evaluation

We employed two popular network traffic datasets, namely Abilene [30] and GÉANT [31], as well as a video dataset TAA [32] to evaluate the performance of the interpolation model. The Abilene dataset consists of a sequence of 12×12 matrices, while GÉANT comprises 23×23 matrices. The TAA dataset is derived from the ending credits of the TV show *The Age of Awakening*, with an image resolution of 176×144. Due to its slow, scroll-like motion, TAA is well suited for frame interpolation tasks. We split each dataset into training, validation, and test sets with a ratio of 80%, 10%, and 10%, respectively. All experiments were conducted on a machine equipped with an Intel ®Core ™i7-10750H CPU, a GeForce RTX 2060 GPU, 16 GB RAM, using the PyTorch framework.

We employ the metrics average error rate (AER) in (2), average NMAE (ANMAE) in (3), and average RMSE (ARMSE) in (4) to evaluate the performance of the TIM model. Smaller values of these metrics indicate that the interpolated TM is closer to the ground truth. Specifically, AER and ANMAE measure the relative error magnitude between X^k and $\hat{X}^k$, whereas ARMSE quantifies their absolute deviation. As relative error metrics, AER and ANMAE are often used to qualitatively assess reconstruction quality. Empirical results reported in existing traffic matrix studies [15, 39, 40] suggest that values below approximately 0.2 are generally associated with high-quality reconstruction, values in the range of [0.2,0.6] indicate moderate or fair performance, while larger errors often render the results less useful for downstream applications.

$$AER\left(\{X^k\}, \{\hat{X}^k\}\right) = \mathbb{E}_k \left[\frac{\|X^k - \hat{X}^k\|_F}{\|X^k\|_F} \right], \tag{2}$$

$$ANMAE\left(\{X^k\}, \{\hat{X}^k\}\right) = \mathbb{E}_k \left[\frac{\left|X^k - \hat{X}^k\right|}{|X^k|} \right], \tag{3}$$

$$ARMSE\left(\{X^k\}, \{\hat{X}^k\}\right) = \mathbb{E}_k \left[\|X^k - \hat{X}^k\|_F \right]. \tag{4}$$

As depicted in Table 1, the TIM model performs significantly better on the TAA dataset than on the two network traffic datasets, Abilene and GÉANT. These results demonstrate that CNN-based TIM can effectively extract features from image frames but performs poorly when extracting features from TMs, even though images and TMs are both organized as matrices.

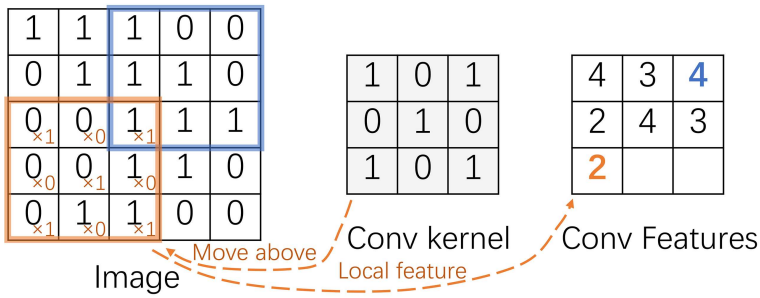

Figure 5 (Color online) Convolution operation of CNN.

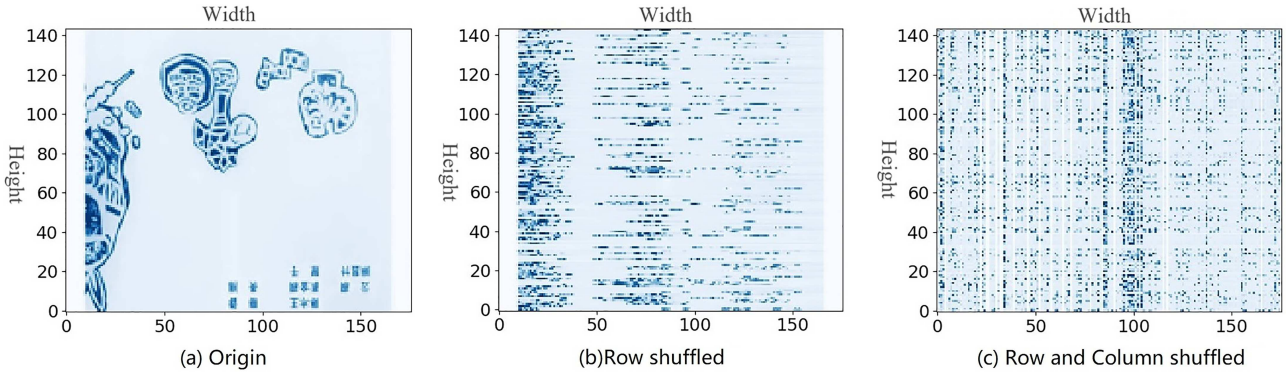

Figure 6 (Color online) (a) Matrix of TAA (snapshot from video 01:05); (b) matrix of TAA after rows are shuffled; (c) matrix of TAA after both rows and columns are shuffled.

5 Why can CNN perform well on images but not on network TMs?

5.1 How does CNN work?

In a CNN, each convolution kernel (i.e., a feature extractor) is applied to extract features from the input matrix through a convolution operation defined as follows:

$$out = kernel \otimes input + bias, \tag{5}$$

where *kernel* represents the weight of the convolution, *input* is the sub-matrix operated on by the convolution kernel, $\otimes$ is a dot-wise product, *bias* is a float number, and *out* in (5) represents the extracted features from the input sub-matrix.

The sub-matrix involved in the convolution operation (e.g., the orange box in Figure 5) is called the "Receptive Field". A convolution operation extracts local features from the "Receptive Field", which are contained in *out*.

CNN's convolution process consists of multiple convolution operations. Figure 5 shows a convolution process with $bias = 0$. The convolution kernel sequentially performs operations, moving from left to right and top to bottom of the input matrix. Each operation extracts a local feature from the corresponding "Receptive Field". For example, when the convolution kernel moves to the left-bottom sub-matrix (Figure 5 orange box), we can obtain the local feature "2" by performing a dot-wise product of the convolution kernel and the sub-matrix in the orange box.

5.2 Why does CNN perform differently on images and TMs?

According to (5) and Figure 5, CNNs can extract local features of the "Receptive Field". We hypothesize that the performance of a CNN is related to the spatial locality of the data in the "Receptive Field". The stronger the locality, the better the CNN performance.

To verify our hypothesis, we conducted experiments involving row shuffling (Figure 6(b)) and row-column shuffling (Figure 6(c)) on the videos in the TAA dataset. Subsequently, we applied the TIM model to process the shuffled data.

Table 2 lists the interpolation performance. After the rows are shuffled, AER and ANMAE of TAA increase and reach 0.40402 and 0.26571, which are 106.9% and 101.8% larger than that of the original TAA. After both rows

Table 2 Performance of TAA vs. shuffled TAA.

	AER	ANMAE	ARMSE
TAA (origin)	0.19522	0.13166	12.1067
Row shuffled	0.40402	0.26571	24.9017
Row&Col shuffled	0.44668	0.30642	27.2623

Table 3 Similarity metrics.

Formula	Description
$SAD(X_p, X_q) = \sum \lvert X_p - X_q \rvert$	Sum of absolute differences (SAD).
$\cos(X_p, X_q) = \frac{X_p \cdot X_q}{\lvert X_p \rvert \cdot \lvert X_q \rvert}$	Cosine similarity (Cos) [28].
$r(X_p, X_q) = \frac{Cov(X_p, X_q)}{\sqrt{Var[X_p] Var[X_q]}}$	Correlation coefficient (CC), where $Cov(X_p, X_q)$ is the covariance of X_p and X_q, and $Var[X_p]$, $Var[X_q]$ are their variances.
$(X_p; X_q) = \sum_{x_p \in X_p} \sum_{x_q \in X_q} P(x_p, x_q) \log \left(\frac{P(x_p, x_q)}{P(x_p) P(x_q)} \right)$	Mutual information (MI), where $P(x_p, x_q)$ is the joint distribution and $P(x_p)$, $P(x_q)$ are marginal distributions.
$SU(X_p; X_q) = 2 \times \left[\frac{I(X_p; X_q)}{H(X_p) + H(X_q)} \right]$, $H(X_p) = -\sum_{x_p \in X_p} P(x_p) \log_2 (P(x_p))$	Symmetrical uncertainty (SU) [41], where $I(X_p; X_q)$ is the MI of X_p and X_q.

and columns are shuffled, AER and ANMAE of TAA increase and reach 0.44668 and 0.30642, which are 128.7% and 132.7% larger than those of the original TAA. Combining the performance results in Tables 1 and 2, we draw the following conclusions.

• **Why does CNN work well on images?** In Table 1, TIM performs well under TAA. This is because adjacent pixels of images have similar colors, thus high correlations and strong spatial locality. In Table 3, TIM performs poorly under the shuffled TAA. This is because row shuffling and column shuffling destroy the locality of the original image and the correlations of adjacent pixels.

• **Why does CNN perform poorly on network TMs?** TMs represent traffic distribution, with rows and columns corresponding to source and destination nodes. Some sources (destinations) share underlying traffic patterns and have similar traffic behaviors; thus TMs should have some spatial correlations. However, TMs are usually ordered by network node IDs. As a result, correlated sources/destinations may not be adjacent, making it hard for CNNs to detect local correlations (i.e., locality) within the data in TMs.

In summary, CNNs can only extract high-level features from input that already have spatial locality. Because TMs have poor spatial locality, CNNs cannot extract their features well. Even though TMs do have some spatial structures, CNNs have difficulty understanding the matrix to effectively extract the features.

6 Rearranging network TMs by similarity

Even though TMs have some spatial structures, local correlations in TMs are weak. Can we strengthen the data locality in TMs so that they can be better understood by CNNs?

From the shuffled TAA, it is easy to obtain the original TAA with good locality by performing the reverse operations of row and column shuffling. Can we shuffle the rows and columns to change the order of rows and columns in a TM such that the matrix exhibits higher correlations (RQ2)? If so, we expect that such a rearranged matrix could be more suitable for CNNs.

To achieve this goal, one possible approach is to place similar rows (columns) next to each other and dissimilar rows (columns) as far away as possible. In this way, the adjacent entries in TMs will be similar, and the locality is therefore enhanced. There are two key issues to address.

• How to calculate the similarities of rows (columns)?

• How to design a rearrangement algorithm to maximize the locality of the matrix?

6.1 Similarity metric

Different similarity metrics will lead to different rearrangements. We should choose suitable similarity metrics to rearrange TMs. For any two matrix rows (columns) X_p and X_q, we can calculate the similarity using five widely used metrics, as shown in Table 3.

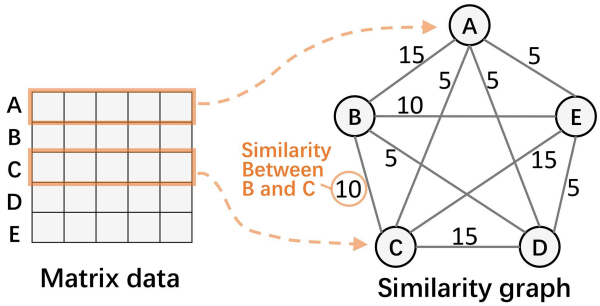

Figure 7 (Color online) The completely weighted undirected graph.

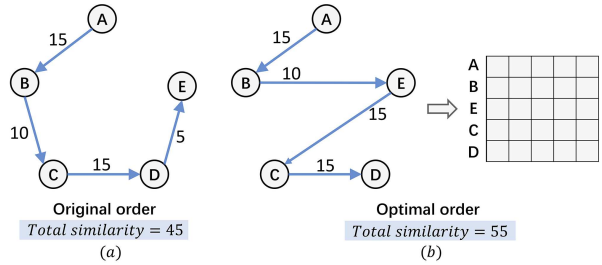

Figure 8 (Color online) (a) The original row order; (b) the optimal row order.

6.2 Rearranging a matrix based on ant colony algorithm

To facilitate matrix rearrangement, we construct two fully weighted undirected graphs, one for row rearrangement and the other for column rearrangement. We will present the graph used for row rearrangement as an example to illustrate our algorithm.

By calculating the similarity between each pair of rows in a matrix, we build a completely weighted undirected graph $G = (V, E)$ to represent the similarity relationship among rows. In the graph, V is the vertex set, with each vertex representing one row in the matrix; E is the edge set connecting the vertices, and the weight of an edge is the similarity between two vertices (thus two corresponding rows of the matrix) of the edge.

Figure 7 gives an example to illustrate the weighted undirected graph. The matrix has 5 rows; hence the graph has 5 vertices. There are 10 edges: A-B, A-C, A-D, A-E, B-C, B-D, B-E, C-D, C-E, D-E, with the edge weight being the number next to the edge. Clearly, one row order of the matrix corresponds to a path in the graph. The original matrix in Figure 7 corresponds to a path A→B→C→D→E (Figure 8(a)) with its total weight being $15 + 10 + 15 + 5 = 45$.

Our goal is to find a rearrangement order that maximizes the similarity in the rearranged matrix. It turns into a traveling salesman problem (TSP), which aims to find a path that travels all nodes with the maximum total weight.

Figure 8(b) shows the TSP solution for the graph in Figure 7. The optimal order is A→B→E→C→D with the total weight being $15+10+15+15 = 55$. We also draw the rearranged matrix in Figure 8(b). Clearly, the similarity level (i.e., 55) in the rearranged matrix is greater than that (i.e., 45) in the original matrix in Figure 7.

However, TSP is a typical NP-complete problem, and its search space increases exponentially with the size of V. The global optimal solution cannot be obtained in polynomial time. The ant colony algorithm is a heuristic global optimization algorithm [42] that can efficiently find an approximate solution in a short time. We design an ant colony-based algorithm to solve the TSP problem.

Given a graph $G = (V, E)$ representing the similarity relationship among rows, the ant colony-based algorithm is executed following the steps below.

Step 1: Set the initial pheromone of each edge on the graph G to 0, and define N_a ants, where $N_a = 1.5 \times |V|$.

Step 2: Randomly select a starting node in V for each ant.

Step 3: Each ant selects a node it has not yet visited. Among all unvisited nodes, the ant can easily select a node with a higher pheromone value from the ant's current node to the node.

Step 4: Repeat Step 3 until all ants pass through all nodes.

Step 5: Each ant adds pheromone to the edge along the path it travels. The smaller the path's total weight (similarity), the less the pheromone added.

Step 6: Record the path with the largest weight. Multiply pheromone on each edge by the evaporation factor.

Step 7: Repeat Steps 2–6 until the stop condition is satisfied: reaching the maximum iteration rounds M.

To rearrange a matrix, we can first run the ant colony-based algorithm on the row graph $G_{row} = (V, E)$ to rearrange the rows, and then run the algorithm on the column graph $G_{column} = (V, E)$ to rearrange the columns.

6.3 Evaluation

We would like to investigate the impact of matrix rearrangement on the TIM model and identify suitable similarity metrics for matrix rearrangement.

• For TMs, we could not find a similarity metric that yields significant interpolation performance gains. Matrix rearrangement based on some metrics even results in worse interpolation performance.

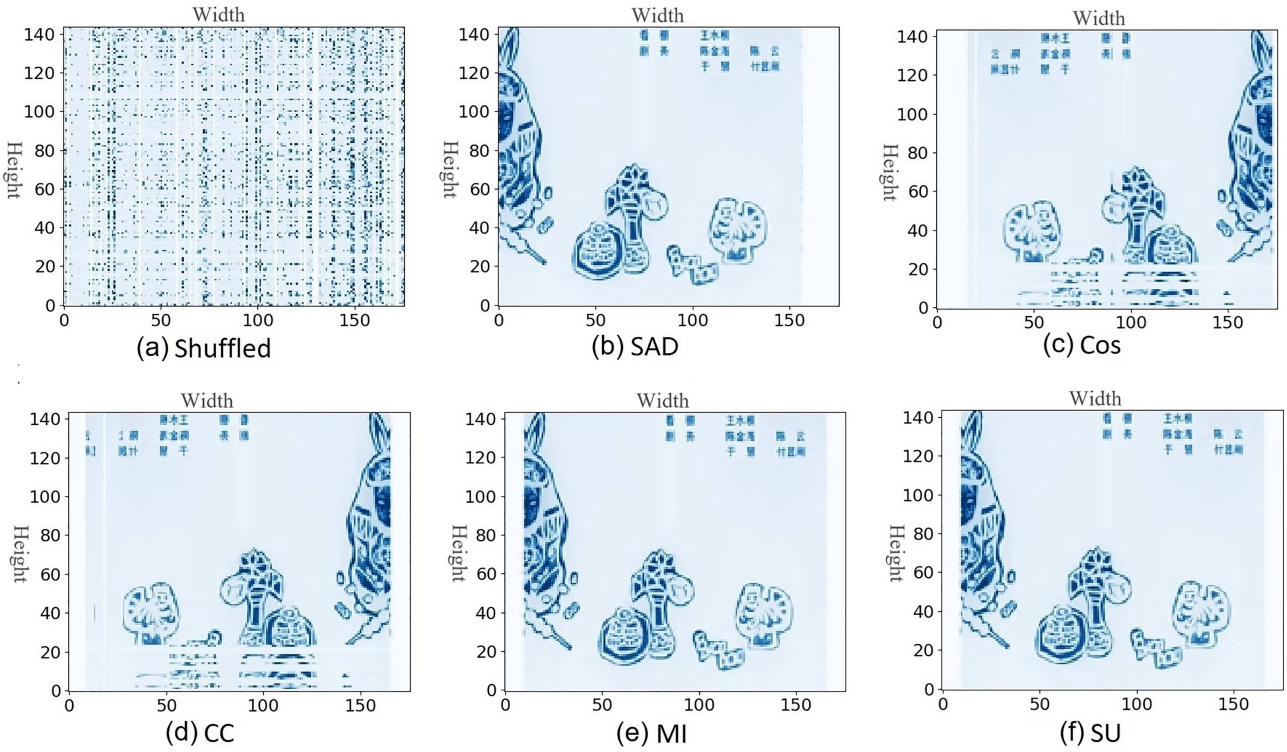

Figure 9 Matrices of TAA after both rows and columns are shuffled, and rearrange the shuffled matrix using the above five similarity metrics.

• For video TAA, we find that the similarity metric SAD provides the best interpolation performance. The interpolation performance under MI and SU is slightly worse than that under SAD. Figure 9 shows the rearranging results of the ant colony algorithm (based on the above five metrics) for the shuffled TAA. It can be seen that the ant colony algorithm perfectly restores the original order (Figure 6(a)) using metrics SAD, MI, and SU.

7 R&E neural network for matrices

From Subsection 6.3, we have learned that matrix rearrangement has some impact on CNN-based models. However, finding a suitable rearrangement scheme is a challenging task.

(1) There are countless metrics to measure similarity, and each results in a different rearrangement order. Although we can evaluate the performance of all metrics, as shown in Table 4, to obtain the optimal one, the cost would be prohibitive.

(2) Existing similarity metrics may not lead to the optimal rearrangement order. In Table 4, among all the five similarity metrics tested, we do not find a metric that is suitable for TMs.

(3) Different datasets may require different similarity metrics because they may possess unique data characteristics. Even if a metric is suitable for one dataset, it may not be appropriate for others. In Table 4, the metric SAD is suitable for video but not for TMs.

This raises a question: is there a way to automatically find a suitable rearrangement scheme based on the characteristics of the data being processed?

7.1 How could a matrix rearrangement be realized?

From the perspective of matrix multiplication, we would like to find a way of rearranging the matrix. More specifically, we want to identify the possible formats of permutation matrices for row and column rearrangements. Figure 10 shows two examples.

• Row rearrangement ($X^1 \to X_r^{'1}$) in Figure 10(a). To obtain the rearranged matrix $X_r^{'1}$, the original matrix X^1 is left-multiplied by a matrix R_r, resulting in the swapping of its first and second rows (Figure 10(a), blue arrows).

• Column rearrangement ($X^2 \to X_c^{'2}$) in Figure 10(b). To obtain the rearranged matrix $X_c^{'2}$, the original matrix X^2 is right multiplied by a matrix R_c, so the first, second, and third columns of $X_c^{'2}$ are the third, first, and second columns of X^2 (Figure 10(b), green arrows).

Table 4 Performance under rearrangement with five similarity metrics (Subsection 6.1). The best results are in bold.

		AER	ANMAE	ARMSE
Abilene	Origin	0.29913	0.29001	0.00204
	SAD	0.32292	0.28319	0.00221
	Cos	0.29448	**0.24981**	0.00201
	CC	**0.27195**	0.26908	**0.0018**
	MI	0.2933	0.26839	0.00198
	SU	0.29526	0.25891	0.00202
GÉANT	Origin	0.22454	**0.2065**	13028.1
	SAD	0.55341	0.42391	32228.2
	Cos	0.24611	0.3276	14441.8
	CC	0.2964	0.2653	17311.6
	MI	0.2741	0.25629	16000.1
	SU	**0.2227**	0.23807	**13016.6**
TAA	Shuffled	0.4466	0.30642	27.2623
	SAD	**0.20133**	**0.12933**	**12.5951**
	Cos	0.24665	0.15417	15.0814
	CC	0.2381	0.15077	14.7266
	MI	0.20551	0.14401	12.6112
	SU	0.2055	0.13205	12.7693

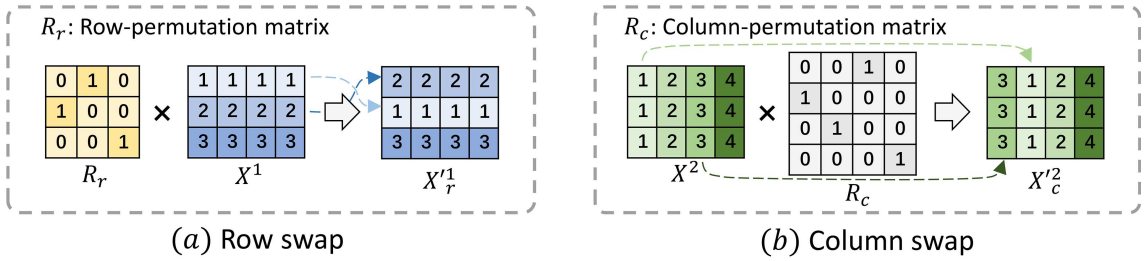

Figure 10 (Color online) Matrix rearrangement through matrix multiplication with permutation matrices.

In these examples, we name R_r and R_c the row-permutation matrix and column-permutation matrix, respectively. A permutation matrix is square, with only one element being 1 in each row and each column, and the remaining elements being 0.

To rearrange a matrix X^c, we derive two principles. (1) By left multiplying the matrix X^c by a row-permutation matrix R_r, we can rearrange the rows in matrix X^c. (2) By right multiplying the matrix X^c by a column-permutation matrix R_c, we can rearrange the columns in matrix X^c.

Given a TM X^c in $\{X^c\}$, after both row and column rearrangements of X^c, we can obtain the matrix $X^{'c}$. The process can be expressed as

$$X^{'c} = R_r \cdot X^c \cdot R_c. \tag{6}$$

Clearly, the positions of element 1 in the permutation matrix determine the final rearranged matrix. Since different permutation matrices correspond to different rearrangement orders, the problem of matrix rearrangement is transformed into the problem of finding the optimal permutation matrices.

7.2 Rearrangement based on binary neural network

To avoid the inefficient manual search of similarity metrics as in Section 6, we leverage the powerful fitting ability of neural networks to automatically learn permutation matrices.

We first design a neural network framework depicted in Figure 11 to automatically learn the permutation matrices and train the TIM model. The framework consists of a Swap layer, a TIM model (Subsection 4.1), and a Restoration layer.

The Swap layer comprises two parameter matrices P_r and P_c, representing row and column permutation matrices R_r and R_c, respectively. After performing row and column rearrangement through the Swap layer, the input $\{X^c\}$ is transformed into the rearranged matrices $\{X^{'c}\}$, and the interpolated matrices $\{\hat{X}^{'k}\}$ are obtained by the TIM

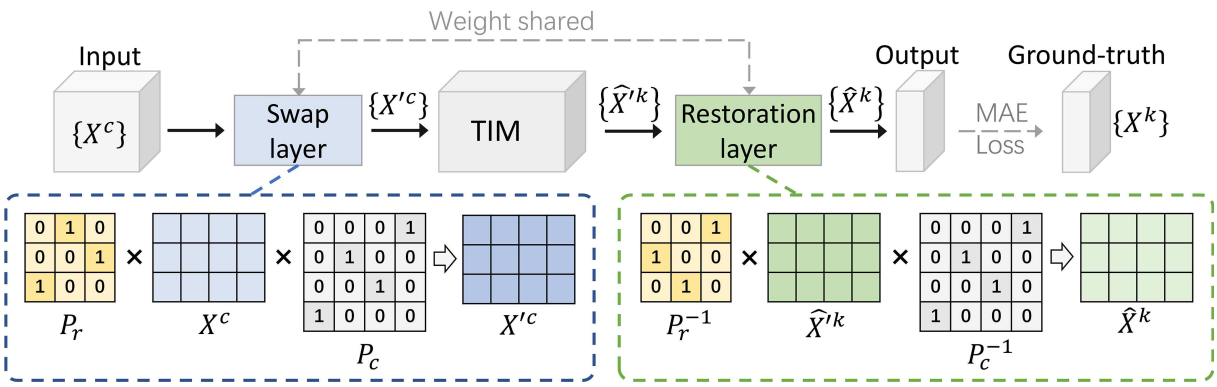

Figure 11 (Color online) A neural network to rearrange matrices for temporal interpolation.

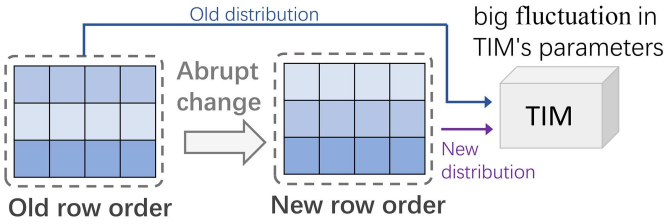

Figure 12 (Color online) "Abrupt change" problem.

model. The Restoration layer can transform the interpolated matrices $\{\hat{X}'^k\}$ back to the matrix's original order (i.e., $\{\hat{X}^k\}$) using the inverse matrices P_r^{-1} and P_c^{-1}. Thus, the output $\{\hat{X}^k\}$ and ground-truth $\{X^k\}$ have the same row and column order. To ensure that the parameter matrix of the Swap layer meets the definition of a permutation matrix, we implement the Swap layer using a binary neural network [43–45].

The binary neural network can make the parameter matrices consist only of numbers 0 or 1 through the "binarizing operation". Let P_{bin} denote the parameter matrix P_r after the binarizing operation. In this way, we can use P_{bin} to multiply the input X^c and update P_r through back-propagation.

However, during implementation, the binary neural network introduces a new problem: "Abrupt change". As illustrated in Figure 12, since permutation matrices are discrete, when the binarized permutation matrix P_{bin} changes, the row order corresponding to the TIM input X'^c will suddenly shift from the old row order (the left matrix in Figure 12) to the new row order (the right matrix in Figure 12). Since the already-trained model parameters are based on the old row order, an "Abrupt change" leads to fluctuations in the model's parameter updates, making it difficult to converge.

7.3 R&E neural network

To address the "Abrupt change" problem, we generalize the concept of the perturbation matrix to the domain $\mathbb{R}$ of real numbers and propose a feasible rearrangement framework, R&E. Specifically, instead of using 0–1 permutation matrices directly, we propose a new WS (weighted swap) layer and a Restoration layer that utilize relaxed weighted permutation matrices.

The WS layer consists of a row-weighted permutation matrix and a column-weighted permutation matrix. Compared with the permutation matrix, the weighted permutation matrix is continuous, and the value range of the matrix elements belongs to the real number field $\mathbb{R}$. Clearly, a permutation matrix in Section 6 is a special case of the weighted permutation matrix defined in this section.

Figure 13 demonstrates the effectiveness of the weighted permutation matrices. X^3 is an input matrix. The darker the blue color, the larger the value. It can be seen that the locality of X^3 is not obvious. W_r (W_c) is a row (column)-weighted permutation matrix. After X^3 is multiplied with row-column weighted permutation matrices, the rearranged matrix X'^3 exhibits a clear locality. This example also shows that the weighted permutation matrices not only rearrange rows and columns but also enhance the locality.

By incorporating the new WS and Restoration layer into the TIM model, we design a new R&E network in Figure 14.

The input is first fed to the WS layer. In the WS layer, after being left-multiplied by the weighted permutation matrix W_r and right-multiplied by the weighted permutation matrix W_c, the input X^c is transformed to X'^c (blue

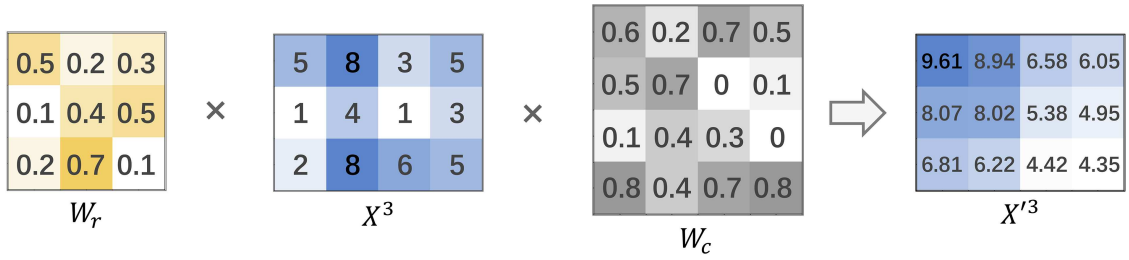

Figure 13 (Color online) Matrix rearrangement and reinforcement through matrix multiplication with weighted permutation matrices.

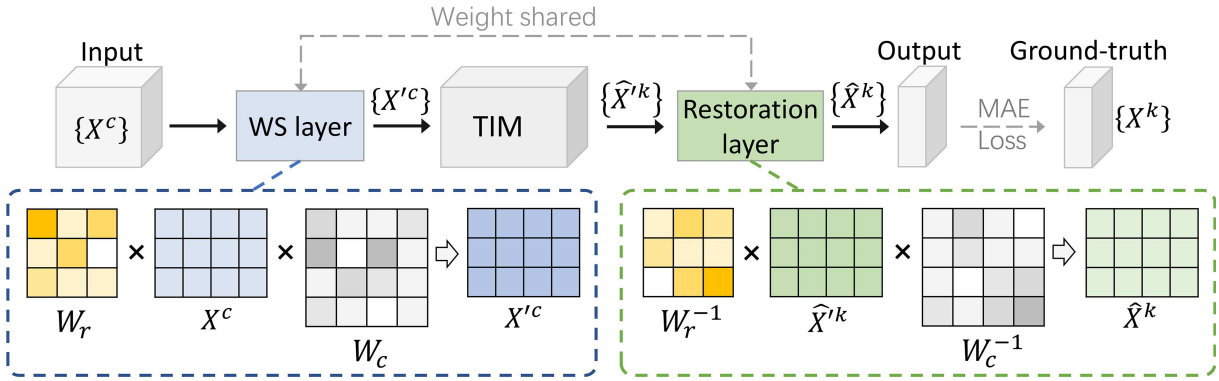

Figure 14 (Color online) A neural network to rearrange and reinforce matrices for temporal interpolation.

dashed box in Figure 14), and the operation can be expressed as

$$X^{'c} = W_r \cdot X^c \cdot W_c. \tag{7}$$

By feeding the transformed $X^{'c}$ to TIM, the interpolated matrices $\{\hat{X}^{'k}\}$ are obtained.

Similar to the Restoration layer in Figure 11, the Restoration layer in Figure 14 uses the inverse matrices W_r^{-1} and W_c^{-1} to transform the interpolated matrices $\{\hat{X}^{'k}\}$ back to the dataset's original order. The operation can be expressed as

$$\hat{X}^k = W_r^{-1} \cdot \hat{X}^{'k} \cdot W_c^{-1}. \tag{8}$$

It should be noted that W_r^{-1} and W_c^{-1} are the inverse matrices of W_r and W_c, respectively. In our design, the WS layer and the Restoration layer share the matrix weights.

7.4 Evaluation

The proposed R&E network is a general framework that can be used to enhance CNN's ability in understanding domain-specific structures of matrix data in various fields. To demonstrate its generality and effectiveness, we test the TIM model's performance not only using TMs corresponding to the datasets Abilene and GÉANT and images in TAA, but also using matrices containing data from a large video dataset called Firework[1)] (with the image size 270×480) and the urban monitoring dataset TaxiBJ [46]. TaxiBJ is composed of two kinds of sequential 32×32 USMs, which respectively record the inflows and outflows of people every 30 min from 1st Nov. 2015 to 10th Apr. 2016 in Beijing, the two kinds of USMs are referred to as "TaxiBJ (In)" and "TaxiBJ (Out)" in Table 5.

In Table 5, "Origin" refers to the original model in Section 4, and "Weighted" refers to the neural network model in Figure 14. We set the maximum number of epochs for training to 200, the optimizer is Adam, and its learning rate $lr = 0.0005$. In Table 5, we observe the following.

• Our Weighted model is very effective in enhancing the locality of TMs, thus facilitating CNN to effectively extract the features hidden in TMs. For both network traffic datasets Abilene and GÉANT, the interpolation accuracy is greatly improved when using the Weighted model. Compared with the Origin model, AER, ANMAE, and ARMSE decrease significantly. For example, for the Abilene dataset, AER, ANMAE, and ARMSE are 0.29913,

1) The video of Firework. https://vimeo.com/100001237.

Table 5 Performance under R&E neural network. The best results are in bold.

		AER	ANMAE	ARMSE
Abilene	Origin	0.29913	0.29001	0.00204
	Weighted	**0.18043**	**0.15830**	**0.00124**
GÉANT	Origin	0.22454	0.2065	13028.1
	Weighted	**0.16639**	**0.1769**	**9774.17**
TAA	Origin	0.19522	0.13166	12.1067
	Weighted	**0.16911**	**0.11257**	**10.2591**
TaxiBJ (In)	Origin	0.23714	0.21107	39.0882
	Weighted	**0.18911**	**0.16411**	**30.9701**
TaxiBJ (Out)	Origin	0.24530	0.20728	40.1566
	Weighted	**0.18895**	**0.16879**	**28.3137**
Firework	Origin	0.39509	0.24013	5.1764
	Weighted	**0.36855**	**0.22751**	**5.0256**

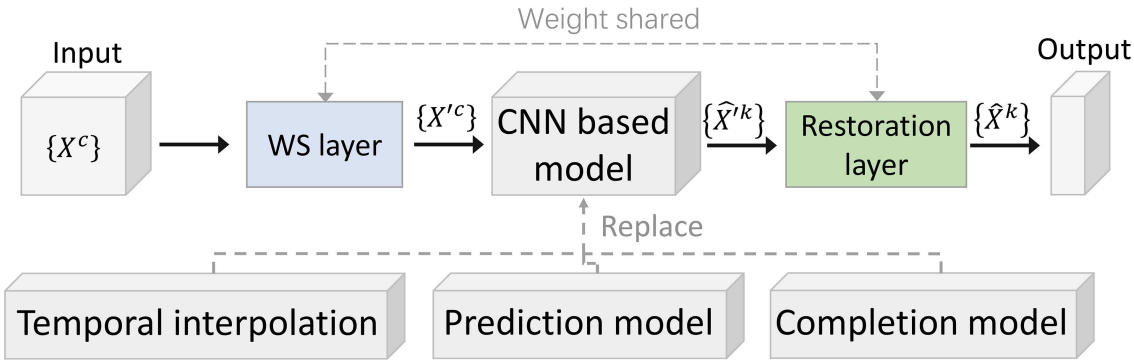

Figure 15 (Color online) The general process of R&E neural network.

0.29001, and 0.00204, respectively, for the Origin model, while they are 0.18043, 0.15830, and 0.00124 in the Weighted model. AER, ANMAE, and ARMSE are nearly halved compared to the Origin model.

• In the case of applying USMs in urban monitoring, the interpolation performance under the Weighted model is obviously improved compared to that under the Origin model in the TaxiBJ dataset, which demonstrates that our Weighted model can also effectively help CNN extract the features hidden in USMs.

• Even for video data, such as TAA and Firework, our Weighted model can further enhance their locality, which helps CNN to enhance its feature extraction ability. For video data TAA, which performs well in the Origin model, the interpolation performance is still slightly improved. However, as shown in Table 4, the best performance under the method of rearrangement with a similarity metric cannot exceed its Origin model.

Therefore, our Weighted model can facilitate CNN to more effectively extract features in matrix data. It is general and can be applied to enhance the locality of various types of matrix data in different research fields.

8 General R&E neural network to enhance CNN's understanding

We expect our R&E network to be a general framework that can enhance CNN's ability to understand domain-specific structures in input matrix data and can also bring significant gains to other applications.

By using other CNN-based models to replace the TIM model in Figure 11, we have the general framework of the process of the R&E network, as shown in Figure 15. In this section, we introduce two more tasks: prediction and missing data completion to evaluate the R&E network.

The prediction model is shown in Figure 16(a). Let X^t denote the matrix collected at time t. Given a historical matrix sequence $\left\{X^{t-T}, ..., X^t\right\}$, where $T = 8$, the prediction task is to predict the matrix of the future time X^{t+1}. We train the whole model in an end-to-end manner using the following MAE loss:

$$MAE(X^{t+1}, \hat{X}^{t+1}) = \left|X^{t+1} - \hat{X}^{t+1}\right|, \tag{9}$$

where $\hat{X}^{t+1}$ and X^{t+1} are the predicted matrix and ground-truth at time $t+1$.

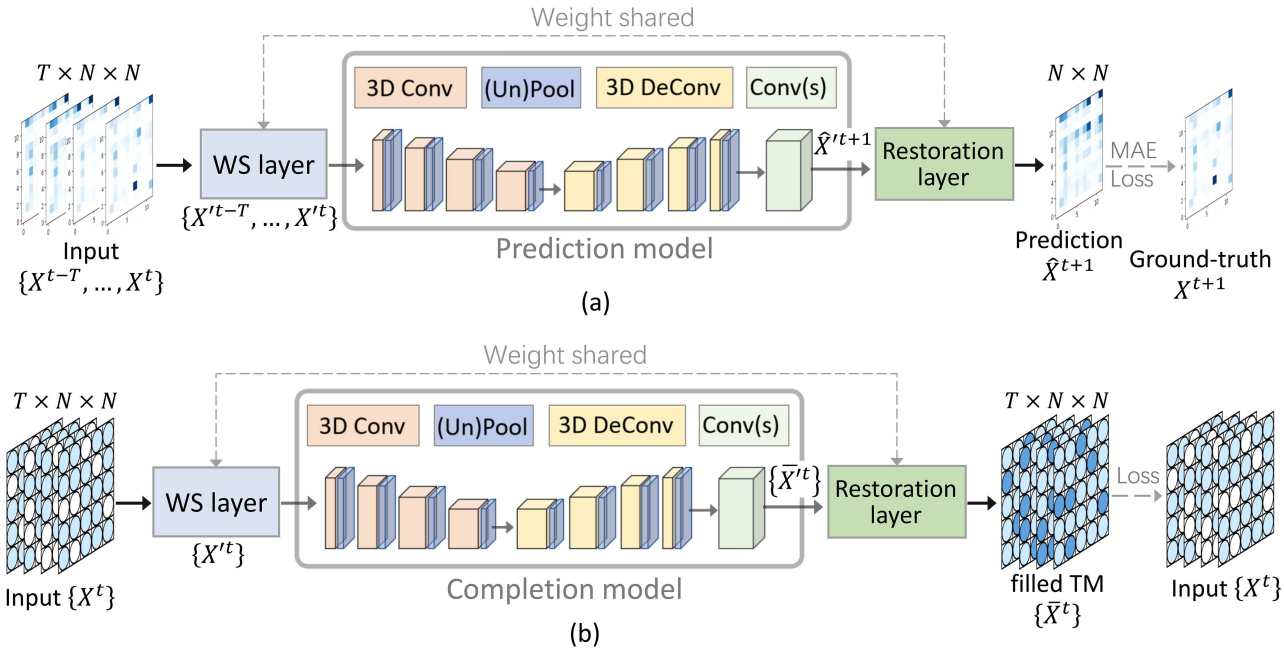

Figure 16 (Color online) (a) Prediction model and (b) missing data completion model with R&E.

Table 6 Performance of prediction task under R&E (with a historical matrix length $T = 8$). The best results are in bold.

		AER	ANMAE	ARMSE	Time (s)
Abilene	Origin	0.3423±0.0054	0.2790±0.0046	0.00208±0.00002	204±5
	Weighted	**0.2521±0.0021**	**0.1899±0.0028**	**0.00155±0.00002**	245±11
	Port (Static)	0.2529±0.0045	0.2032±0.0037	0.00156±0.00002	235±6
	Port (Dynamic) 100 epochs	0.2644±0.0039	0.2140±0.0025	0.00163±0.00002	**131±3**
GÉANT	Origin	0.2937±0.0039	0.3094±0.0039	16427±192	286±11
	Weighted	**0.2160±0.0018**	**0.2141±0.0023**	**12143±150**	**342±17**
	Port (Static)	0.2263±0.0033	0.2236±0.0026	12640±193	329±13
	Port (Dynamic) 100 epochs	0.2428±0.0038	0.2417±0.0043	13570±157	165±7
TAA	Origin	0.2864±0.0045	0.1788±0.0019	15.8159±0.1642	2365±68
	Weighted	**0.2406±0.0034**	**0.1490±0.0016**	**13.1059±0.1471**	2655±79
	Port (Static)	0.2520±0.0038	0.1569±0.0026	13.8305±0.245	2503±74
	Port (Dynamic) 100 epochs	0.2548±0.0045	0.1572±0.002	13.8899±0.2209	**1407±35**
TaxiBJ (In)	Origin	0.2598±0.0046	0.2411±0.0021	38.7693±0.6250	422±14
	Weighted	0.2137±0.0037	0.1842±0.0029	32.4836±0.2849	530±24
	Port (Static)	**0.2036±0.0028**	**0.1828±0.0034**	**31.3336±0.5670**	489±13
	Port (Dynamic) 100 epochs	0.2128±0.003	0.1920±0.0034	31.7452±0.3087	**249±9**
TaxiBJ (Out)	Origin	0.2470±0.0025	0.2332±0.0042	37.4700±0.4113	413±11
	Weighted	**0.2059±0.0032**	**0.182±0.0024**	**30.3993±0.2802**	463±22
	Port (Static)	0.2122±0.0022	0.1925±0.0035	31.6221±0.3418	428±10
	Port (Dynamic) 100 epochs	0.2172±0.0033	0.1877±0.003	32.7943±0.3270	**237±6**
Firework	Origin	0.5492±0.0095	0.3307±0.0041	7.4340±0.0629	10980±272
	Weighted	**0.5069±0.0077**	**0.3079±0.0055**	**6.9911±0.0687**	14594±590
	Port (Static)	0.5443±0.0097	0.3331±0.003	7.3529±0.0782	13262±505
	Port (Dynamic) 100 epochs	0.5425±0.0059	0.3149±0.0043	7.4267±0.1072	**7473±256**

The missing data completion model is shown in Figure 16(b). Given a sparse matrix sequence $\{X^t\}$ with only partial entries being observed, missing data completion is performed to infer the unobserved entries (refer to [15]). Due to the high cost of performing traffic measurements between each source-destination pair, network TMs are

Table 7 Performance of missing data completion task under R&E (50% sampling ratio). The best results are in bold.

		AER	ANMAE	ARMSE	Time (s)
Abilene	Origin	0.3478±0.0043	0.3018±0.0054	0.00048±0.00001	253±11
	Weighted	**0.2807±0.0039**	**0.2166±0.0033**	**0.00035±0.00001**	286±14
	Port (Static)	0.2956±0.0055	0.2444±0.0024	0.00039±0.00001	261±12
	Port (Dynamic) 100 epochs	0.3013±0.0038	0.237±0.0036	0.00038±0.00001	**141±6**
GÉANT	Origin	0.3522±0.0033	0.401±0.0039	3077±46	388±17
	Weighted	**0.2824±0.0033**	**0.2793±0.0037**	**2146±33**	456±18
	Port (Static)	0.3058±0.0042	0.3482±0.0052	2670±29	445±20
	Port (Dynamic) 100 epochs	0.3141±0.0057	0.3207±0.0033	2445±34	**231±11**
Seattle	Origin	0.8458±0.0137	0.6771±0.0098	0.1220±0.0014	2538±74
	Weighted	0.8136±0.0094	**0.556±0.0061**	**0.1012±0.0015**	2604±131
	Port (Static)	**0.8046±0.0078**	0.5667±0.0079	0.1035±0.0010	2686±56
	Port (Dynamic) 100 epochs	0.8087±0.0106	0.5723±0.0064	0.1041±0.0015	**1272±43**
TAA	Origin	0.3046±0.0054	0.1904±0.0022	16.6589±0.1410	7245±229
	Weighted	**0.2663±0.0046**	**0.1633±0.0028**	**14.3798±0.2337**	7859±381
	Port (Static)	0.2839±0.0032	0.1705±0.0025	15.3548±0.1884	7083±349
	Port (Dynamic) 100 epochs	0.2748±0.0029	0.1708±0.0029	15.2337±0.1293	**3817±92**
TaxiBJ (In)	Origin	0.2671±0.0027	0.2521±0.0041	11.7253±0.1389	691±22
	Weighted	**0.1836±0.0025**	**0.1705±0.0019**	**7.9932±0.0880**	760±18
	Port (Static)	0.2015±0.0039	0.1857±0.0025	8.7346±0.0808	727±16
	Port (Dynamic) 100 epochs	0.1901±0.0021	0.1772±0.0025	8.3506±0.0925	**383±10**
TaxiBJ (Out)	Origin	0.2712±0.0027	0.2467±0.0038	11.6235±0.1331	697±33
	Weighted	**0.1848±0.0027**	**0.172±0.0032**	**8.2135±0.0741**	723±21
	Port (Static)	0.2208±0.0027	0.208±0.003	9.6748±0.1246	717±21
	Port (Dynamic) 100 epochs	0.1881±0.0016	0.1749±0.0017	8.2790±0.1016	**363±17**
Firework	Origin	0.5275±0.0064	0.3064±0.0042	0.5368±0.0098	40782±1858
	Weighted	**0.5097±0.0059**	0.3071±0.0036	**0.5310±0.0044**	41923±1190
	Port (Static)	0.5247±0.0052	0.3043±0.0034	0.5339±0.0048	40868±1814
	Port (Dynamic) 100 epochs	0.5331±0.0084	**0.302±0.0035**	0.5383±0.0100	**1996±50**

usually sparse with only a subset of entries observed. We train the whole model in an end-to-end manner using the following MAE_{obs} loss:

$$MAE_{obs}\left(\left\{\bar{X}^t\right\},\left\{X^t\right\}\right)=\mathbb{E}_t\left[\mathrm{Mask}_{X^t}\odot\left|\bar{X}^t-X^t\right|\right], \tag{10}$$

where X^t is the model input, $\bar{X}^t$ is the output with all matrices completed, $\mathbb{E}_t$ denotes expectation, Mask_{X^t} is the set of observed positions of X^t, and $\odot$ is the dot product.

Tables 6 and 7 present the performance of the two tasks. The "Weighted" model refers to the R&E network enhanced model shown in Figures 16 and 10, while the "Origin" model denotes the corresponding original model. Two portable strategies are evaluated: "Port (Static)" and "Port (Dynamic)". In "Port (Static)", the permutation matrices are directly initialized using those learned from the traffic temporal interpolation task in Subsection 7.4, without further retraining. "Port (Dynamic)" adopts the same initialization but retrains the permutation matrices during task-specific learning. For the missing data completion task (Table 7), the sampling ratio (i.e., the proportion of observed entries) is set to 50%. As model performance generally converges within 200 epochs, the maximum number of epochs is set to 200 for the "Origin", "Weighted", and "Port (Static)" models. Since "Port (Dynamic)" converges faster, its training is limited to 100 epochs. All other settings follow Subsection 7.4. Each experimental scenario is repeated 5 times, with mean performance and standard deviation reported using the ± notation. Table 7 further includes the Seattle [47] dataset, consisting of 699 network latency matrices of size 99×99. Owing to its limited scale, this dataset is widely used for missing data completion [40,48] rather than prediction or interpolation tasks, which typically require a larger amount of data in the matrix. By combining Tables 6 and 7 with Table 5, we observe the following.

• The R&E network is a general network that performs well in different tasks, enhancing CNN's understanding and maximizing its benefits. The performance of the Weighted model is much better than that of the Origin model

across all three base tasks.

• The R&E network demonstrates strong portability. The permutation matrices learned from one task can be easily applied to other tasks, achieving good performance with less training time. The accuracy of the Portable models (both Port (Static) and Port (Dynamic)) is only slightly lower than that of the Weighted model, but still much higher than that of the Origin model. With permutation matrices transferred from other tasks, the Port (Dynamic) model, trained for only 100 epochs, achieves accuracy comparable to the Weighted and Port (Static) models trained for 200 epochs. Furthermore, the training time of Port (Dynamic) is nearly half that of the Weighted model (200 epochs).

To evaluate the time overhead from R&E's inference and backward stages, we use Abilene ('Weighted') in Table 6 as an example. Using torch.cuda.Event and register_backward_hook, we record the additional average time per batch as 0.1208 and 0.1496 s (44.67% vs. 55.32%), indicating that the backward incurs higher overhead than inference.

9 Conclusion

CNNs are well-suited for extracting high-level features from inputs with strong spatial locality. While TMs exhibit some spatial structure, we find that CNNs perform well on images but poorly on TMs due to the low spatial locality in TMs.

To address this, we explored several methods to reformulate TMs for better compatibility with CNNs. First, we proposed an ant colony-based algorithm to rearrange the matrix using five similarity metrics. While one metric proved effective for images, none were suitable for TMs. To overcome this, we designed a novel R&E network to learn permutation matrices that rearrange TMs and improve their spatial locality. Experimental results on three TM-based tasks—temporal interpolation, prediction, and missing data completion—demonstrate that the R&E network significantly enhances the locality of TMs, maximizing the benefits of CNNs.

As DL continues to advance rapidly, more network problems can be addressed using DL techniques. However, the unique characteristics of networks pose challenges to fully leveraging DL's potential in this domain. We hope that our work, as a case study, inspires further research and reflection in the networking community to maximize the impact of DL in solving network problems.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 62321166652, 62025201), Project of Yuelushan Center for Industrial Innovation (Grant No. 2025YCII0125), and Hunan Provincial Natural Science Foundation of China (Grant No. 2024JJ3014).

Supporting information Appendixes A and B. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Chen H, Miao Y, Chen L, et al. Software-defined network assimilation: bridging the last mile towards centralized network configuration management with Nassim. In: Proceedings of the ACM SIGCOMM 2022 Conference, 2022. 281–297
2 Rashelbach A, Rottenstreich O, Silberstein M. Scaling open vSwitch with a computational cache. In: Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 2022. 1359–1374
3 Rusek K, Suarez-Varela J, Almasan P, et al. RouteNet: leveraging graph neural networks for network modeling and optimization in SDN. IEEE J Sel Areas Commun, 2020, 38: 2260–2270
4 Fu S, Gupta S, Mittal R, et al. On the use of ML for blackbox system performance prediction. In: Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation, 2021. 763–784
5 Zhang J, Zeng C, Zhang H, et al. LiteFlow: towards high-performance adaptive neural networks for kernel datapath. In: Proceedings of the ACM SIGCOMM 2022 Conference, 2022. 414–427
6 Piet J, Nwoji D, Paxson V. GFast: automating generation of flexible network traffic classifiers. In: Proceedings of the ACM SIGCOMM 2023 Conference, 2023. 850–866
7 Tune P, Roughan M, Haddadi H, et al. Internet Traffic Matrices: a Primer. New York: Association for Computing Machinery, 2013. 1–56
8 Gui F, Wang S, Li D, et al. RedTe: mitigating subsecond traffic bursts with real-time and distributed traffic engineering. In: Proceedings of the ACM SIGCOMM 2024 Conference, 2024. 71–85
9 Chabarek J, Sommers J, Barford P, et al. Power awareness in network design and routing. In: Proceedings of the 27th Conference on Computer Communications, 2008. 457–465
10 Gao K, Li D, Chen L, et al. Incorporating intra-flow dependencies and inter-flow correlations for traffic matrix prediction. In: Proceedings of the 28th International Symposium on Quality of Service (IWQoS), 2020. 1–10
11 Gao K, Li D, Chen L, et al. Predicting traffic demand matrix by considering inter-flow correlations. In: Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020. 165–170

12 Xie K, Li X, Wang X, et al. On-line anomaly detection with high accuracy. IEEE ACM Trans Netwing, 2018, 26: 1222–1235

13 Bega D, Gramaglia M, Fiore M, et al. DeepCog: cognitive network management in sliced 5G networks with deep learning. In: Proceedings of IEEE Conference on Computer Communications, 2019. 280–288

14 Khandaker K, Trossen D, Yang J, et al. On-path vs off-path traffic steering, that is the question. In: Proceedings of the ACM SIGCOMM Workshop on Future of Internet Routing & Addressing, 2022. 37–42

15 Xie K, Lu H, Wang X, et al. Neural tensor completion for accurate network monitoring. In: Proceedings of IEEE Conference on Computer Communications, 2020. 1688–1697

16 Wu Q, Chen X, Zhou Z, et al. Deep reinforcement learning with spatio-temporal traffic forecasting for data-driven base station sleep control. IEEE ACM Trans Netwing, 2021, 29: 935–948

17 Xie Y, Jia J, Wen C, et al. Multi-topology contrastive graph representation learning. Sci China Inf Sci, 2026, 69: 122102

18 Zhang X, Gong Y, Zhang X, et al. Mask-and contrast-enhanced spatio-temporal learning for urban flow prediction. In: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023. 3298–3307

19 Ji J, Wang J, Huang C, et al. Spatio-temporal self-supervised learning for traffic flow prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2023. 4356–4364

20 Wang D, Liu Y J, Song B. A credible traffic prediction method based on self-supervised causal discovery. Sci China Inf Sci, 2024, 67: 152303

21 Dong C, Xiong X X, Xue Q L, et al. A survey on the network models applied in the industrial network optimization. Sci China Inf Sci, 2024, 67: 121301

22 Yao K L, Li W J. Re-quantization based binary graph neural networks. Sci China Inf Sci, 2024, 67: 172101

23 Liu Y, Hu T, Zhang H, et al. iTransformer: inverted transformers are effective for time series forecasting. In: Proceedings of the Twelfth International Conference on Learning Representations, 2024

24 Nie Y, Nguyen N H, Sinthong P, et al. A time series is worth 64 words: long-term forecasting with transformers. In: Proceedings of the Eleventh International Conference on Learning Representations, 2023

25 Wang Z, Bai Y, Zhou Y, et al. Can CNNs be more robust than transformers? In: Proceedings of the Eleventh International Conference on Learning Representations, 2023

26 Zeng A, Chen M, Zhang L, et al. Are transformers effective for time series forecasting? In: Proceedings of the AAAI Conference on Artificial Intelligence, 2023. 11121–11128

27 Chakrabarti D, Papadimitriou S, Modha D S, et al. Fully automatic cross-associations. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery And Data Mining, 2004. 79–88

28 Shraga R, Gal A, Roitman H. ADnEV: cross-domain schema matching using deep similarity matrix adjustment and evaluation. Proc VLDB Endow, 2020, 13: 1401–1415

29 Wang S, He L, Cao B, et al. Structural deep brain network mining. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017. 475–484

30 The Abilene observatory data collections. 2023. https://www.cs.utexas.edu/ yzhang/research/AbileneTM/

31 Uhlig S, Quoitin B, Lepropre J, et al. Providing public intradomain traffic matrices to the research community. ACM Spec Interest Group on Data Commu, 2006, 36: 83–86

32 Li T, Zhang Y, Liu G. The ending credits of the age of awakening source. 2021. https://tv.cctv.com/2021/02/02/ vidajnnhiuxbyykl-tos0zo2k210202.shtml

33 Niklaus S, Mai L, Liu F. Video frame interpolation via adaptive convolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017. 670–679

34 Li S, Xu X, Pan Z, et al. Quadratic video interpolation for VTSR challenge. In: Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 2019. 3427–3431

35 Lee H, Kim T, Chung T, et al. AdaCof: adaptive collaboration of flows for video frame interpolation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020. 5316–5325

36 Cheng X, Chen Z. Video frame interpolation via deformable separable convolution. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2020. 10607–10614

37 Zolfaghari M, Ghanei-Yakhdan H, Yazdi M. Real-time object tracking based on an adaptive transition model and extended Kalman filter to handle full occlusion. Vis Comput, 2020, 36: 701–715

38 Kalluri T, Pathak D, Chandraker M, et al. FLAVR: flow-agnostic video representations for fast frame interpolation. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023. 2071–2082

39 Xie K, Xie R, Wang X, et al. NMMF-Stream: a fast and accurate stream-processing scheme for network monitoring data recovery. In: Proceedings of the IEEE Conference on Computer Communications, 2022. 2218–2227

40 Tian J, Xie K, Wang X, et al. TensorMon: a breakthrough in sparse data gathering leveraging tensor-enhanced techniques for system and network monitoring. IEEE Trans Knowl Data En, 2025, 37: 6708–6722

41 Dong S, Ding W, Chen L. Measure correlation analysis of network flow based on symmetric uncertainty. Ksii Trans Internet Inf, 2012, 6: 1649–1667

42 Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B, 1996, 26: 29–41

43 Hubara I, Courbariaux M, Soudry D, et al. Binarized neural networks. In: Proceedings of the Advances in Neural Information Processing Systems, 2016

44 Rastegari M, Ordonez V, Redmon J, et al. XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Proceedings of European Conference on Computer Vision, 2016. 525–542

45 Han K, Wang Y, Xu Y, et al. Training binary neural networks through learning with noisy supervision. In: Proceedings of the International Conference on Machine Learning, 2020. 4017–4026

46 Zhang J, Zheng Y, Qi D. Deep spatio-temporal residual networks for citywide crowd flows prediction. In: Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence, 2017

47 Cappos J, Beschastnikh I, Krishnamurthy A, et al. Seattle: a platform for educational cloud computing. In: Proceedings of the 40th ACM Technical Symposium on Computer Science Education, 2009. 111–115

48 Deng L, Liu X Y, Tsang D H. Real-time network latency estimation with pretrained generative models. IEEE Trans Neur Net Lear, 2025, 36: 18075–18088