Fast and Accurate Tensor Completion with Total Variation Regularized Tensor Trains

Ko, Ching Yun; Batselier, Kim; Daniel, Luca; Yu, Wenjian; Wong, Ngai

DOI
10.1109/TIP.2020.2995061

Publication date
2020

Document Version
Final published version

Published in
IEEE Transactions on Image Processing

Fast and Accurate Tensor Completion With Total Variation Regularized Tensor Trains

Ching-Yun Ko, Kim Batselier, Luca Daniel, *Senior Member, IEEE*, Wenjian Yu, *Senior Member, IEEE*, and Ngai Wong, *Senior Member, IEEE*

***Abstract*— We propose a new tensor completion method based on tensor trains. The to-be-completed tensor is modeled as a low-rank tensor train, where we use the known tensor entries and their coordinates to update the tensor train. A novel tensor train initialization procedure is proposed specifically for image and video completion, which is demonstrated to ensure fast convergence of the completion algorithm. The tensor train framework is also shown to easily accommodate Total Variation and Tikhonov regularization due to their low-rank tensor train representations. Image and video inpainting experiments verify the superiority of the proposed scheme in terms of both speed and scalability, where a speedup of up to 155× is observed compared to state-of-the-art tensor completion methods at a similar accuracy. Moreover, we demonstrate the proposed scheme is especially advantageous over existing algorithms when only tiny portions (say, 1%) of the to-be-completed images/videos are known.**

***Index Terms*— Tensor completion, tensor-train decomposition, total variation, image restoration.**

I. Introduction

TENSORS are a higher-order generalization of vectors and matrices and have found widespread applications due to their natural representation of real-life multi-way data, such as images and videos [1]–[5]. Tensor completion generalizes the matrix completion problem, which aims at estimating missing entries from partially revealed data. For example, grayscale images are matrices (two-way tensors) that are indexed by two spatial variables, while color images are essentially three-way tensors with one additional color dimension. Grayscale and color videos are extensions of grayscale and color images by adding one temporal index. Thus image/video recovery tasks are indeed tensor completion problems. Although one can always regard a tensor completion task as multiple matrix completion problems, state-of-the-art matrix completion algorithms such as [6] have rather high computational costs and poor scalability. Moreover, the application of matrix completion methods to tensorial data overlooks a key insight in tensor completion, namely, the low tensor-rank assumption [4], [7]–[9] inherent to the data. For example, normally every two adjacent frames of a video are shot with a very short time interval, implying that only limited changes are allowed between two adjacent video frames. Similarly, the values among neighboring pixels in an image usually vary slowly. These intuitive low rank ideas have been successfully utilized in research areas such as collaborative filtering [10], multi-task learning [11], image/video recovery [4], [5], [7] and text analysis [12], [13].

A. Related Work

Most existing tensor completion methods are generalizations of matrix completion methods. Traditional matrix completion problems are generally formulated into the construction of a structurally low-rank matrix $\boldsymbol{E}$ that has the same observed entries:

$$\min_{\boldsymbol{E}} \operatorname{rank}(\boldsymbol{E}), \quad \text{s.t. } (\boldsymbol{E}-\boldsymbol{O})_{\Omega}=\mathbf{0},$$

where $\boldsymbol{O}$ represents the observed matrix with zero fillings at the missing entries, and $\boldsymbol{\Omega}$ is the mapping that specifies the locations of known entries. Directly solving the above optimization problem is NP-hard, which resulted in extensive research on solving alternative formulations. Two popular candidates are to minimize the nuclear norm (being the convex envelope of the matrix rank-operator [14], [15]), or to use a factorization method [16] that decomposes the matrix $\boldsymbol{E}$ as the product of two small matrices. The nuclear norm approach has been generalized to tensor completion problems by unfolding the tensor along its k modes into k matrices and summing over their nuclear norms [4], [12], [17]. Total variation (TV) terms have also been integrated into this method in [18]. Correspondingly, the factorization method has also been generalized to tensors [7], [19], [20].

Manuscript received November 11, 2018; revised December 8, 2019 and February 27, 2020; accepted May 3, 2020. Date of publication May 21, 2020; date of current version July 8, 2020. This work was supported in part by the General Research Fund of the Hong Kong Research Grants Council under Project 17246416, in part by the MIT-IBM Watson AI Lab, in part by the MIT Quest Programs, in part by the NSF under Grant R01 EB006847, in part by the Tsinghua University Initiative Scientific Research Program, and in part by the NSFC under Grant 61872206. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Damon M. Chandler. *(Corresponding author: Ching-Yun Ko.)*

Ching-Yun Ko was with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong. She is now with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: cyko@mit.edu).

Luca Daniel is with the Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: luca@mit.edu).

Kim Batselier is with the Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: k.batselier@tudelft.nl).

Wenjian Yu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yu-wj@tsinghua.edu.cn).

Ngai Wong is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: nwong@eee.hku.hk).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TIP.2020.2995061

Another way to tackle tensor completion is based on the tensor multi-rank and tubal rank [21]–[23], which are intrinsically defined on three-way tensors such as color images and grayscale videos. It is remarked that multi-rank and tubal rank inspired methods are only applicable when the data are compressible in a t-SVD representation. Methods that exploit tensor decomposition formats were also introduced to tensor completion problems in recent years. In [24] and [25], the authors use the Canonical Polyadic (CP) decomposition for Bayesian tensor estimators. Imaizumi *et al.* [26] adopted tensor trains (TTs) and solved the optimization problem using a TT Schatten norm [27] via the alternating direction method of multipliers. Bengua *et al.* [28] also combined TT rank optimization with factorization methods introduced in [7]. In [9], Grasedyck *et al.* borrowed the traditional tensor completion problem formulation and adopted the tensor train format as the underlying data structure. By updating slices of the tensor cores using parts of the observed entries, the tensor train is completed through alternating least square problems. Wang *et al.* [29] designed a tensor completion algorithm by expanding the tensor trains in [9] to uniform TT-rank tensor rings using random normal distributed values, which yields higher recovery accuracies. However, using tensor rings suffers from two drawbacks: the relatively large storage requirement of a uniform TT-rank structure and a sensitivity of the obtained solution to its random initialization.

B. Our Contribution

In this paper we propose to adopt tensor trains when performing tensor completion, rather than using the CP and Tucker decomposition. The motivation behind this lies in the fact that determining the CP-rank is NP-hard while the TT-rank is easily determined from an SVD [30]. Also, the Tucker form requires exponentially large storage, which is not as economic as a tensor train. We further reformulate the problem as a regression task. The unknown completed tensor $\boldsymbol{\mathcal{A}}$ is thereby interpreted as an underlying regression model, while the observed "inputs" and "outputs" of the model are the multi-indices (coordinates) and values of the known tensor entries, respectively. The tensor completion problem is then solved by the following optimization problem

$$\min_{\boldsymbol{\mathcal{A}}\in\mathcal{S}_{\mathrm{TT}}^{(d)}} ||\boldsymbol{S}^T \operatorname{vec}(\boldsymbol{\mathcal{A}}) - \boldsymbol{y}||_2^2,$$
$$\text{s.t. } \boldsymbol{\mathcal{A}} \in \mathcal{S}_{\mathrm{TT}}^{(d)}, \quad \text{and TT-rank}(\boldsymbol{\mathcal{A}}) = (R_1, R_2, \ldots, R_d),$$

where $\mathcal{S}_{\mathrm{TT}}^{(d)}$ denotes the subspace of d-order tensor where each element can be represented in the tensor train format with d TT-cores. The binary matrix $\boldsymbol{S}^T$ selects the known entries of $\operatorname{vec}(\boldsymbol{\mathcal{A}})$, where $\operatorname{vec}(\boldsymbol{\mathcal{A}})$ denotes the vectorization of the unknown completed tensor $\boldsymbol{\mathcal{A}}$. The vector $\boldsymbol{y}$ contains the values of the observed tensor entries. Minimizing $||\boldsymbol{S}^T \operatorname{vec}(\boldsymbol{\mathcal{A}}) - \boldsymbol{y}||_2^2$ therefore enforces that the desired solution needs to have nearly the same observed tensor entries. To regularize the problem, an additional low-rank constraint is added together with the requirement that the desired tensor is represented in the tensor train format, which will be explained in Section III. Moreover, other regularization terms such as Total Variation (TV) and Tikhonov regularization are readily integrated into this tensor train framework. The above problem is solved using an iterative method called the alternating linear scheme that iteratively solves small linear systems. The computationally most expensive steps are a singular value decomposition and QR decomposition. The numerical stability and monotonic convergence of the proposed method is guaranteed by these orthogonal matrix factorizations [31], [32].

The flexibility of our proposed model naturally permits various variants and creates more possibilities for recovery tasks. Particularly, the inputs and outputs in the proposed model can be grouped into batches. This favors parallelization and allows more room in tuning the number of equations and unknowns in least square problems. For example, the updating scheme in [9], [29] can be incorporated as one specific variant of our proposed plain architecture (without TV/ Tikhonov regularizers) by grouping the inputs/outputs into I_K batches when updating the k-th core. However, this way of grouping inputs/outputs shows no evidence of balancing the number of equations and unknowns in the resulting least square problems and [9] has demonstrated a consistently inferior performance than [29], while we will show in later experiments that the proposed plain TTC model outperforms [29] in both time and accuracy. Lastly, it is remarked that there is no direct TT format employed in [28]. Instead, the TT rank optimization problem formed refers to enforcing the matricizations along k modes have pre-defined rank $= R_k$. This is equivalent to updating the products of the first k tensor train cores and the last $d-k$ cores as two matrices at the same time via a standard factorization method in matrix completions. In summary, the main contributions of this article are:

- The tensor completion problem is rephrased as a regression task and solved using the alternating linear scheme.
- Both the selection matrix $\boldsymbol{S}$ and tensor $\boldsymbol{\mathcal{A}}$ are never explicitly formed, which lifts the curse of dimensionality and results in a computationally efficient tensor completion algorithm.
- A deterministic TT-initialization method for image and video data is provided, which guarantees consistent results and is demonstrated to speed up convergence of our proposed tensor completion algorithm.
- Both TV and Tikhonov regularization are integrated into the tensor train framework. The addition of these regularization terms comes at almost no additional computational cost. The low TT-rank property of the matrices involved in the TV regularization, especially, ensures a computationally efficient algorithm. To the best of our knowledge, this is the first time that the Total Variation regularization term is incorporated in tensor train form.
- We propose a factorization scheme of the physical indices that can be exploited through tensor trains, which results in a better scalability of our method compared to state-of-the-art methods.
- The efficacy of the proposed algorithm is demonstrated by extensive numerical experiments. The proposed method shows a particular effectiveness and efficiency ($155\times$ speedup) in recovering severely damaged (large portions of missing pixels) high-resolution images. A comparable

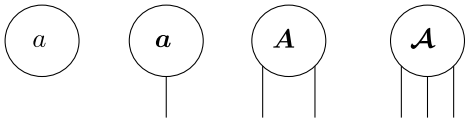


Fig. 1. Graphical depiction of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$ and 3-way tensor $\boldsymbol{\mathcal{A}}$.

performance to the state-or-the-art methods is demonstrated for small image inpainting tasks.

The outline of the article is as follows. Necessary tensor and tensor train preliminaries are briefly introduced in Section II. The proposed tensor train completion model and algorithm are discussed in Section III. Numerical experiments comparing our proposed algorithm with state-of-the-art methods are given in Section IV. Finally, conclusions are drawn in Section V.

II. Preliminaries

A d-way or d-order tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is an array where each entry is indexed by d indices $i_1, i_2, \ldots, i_d$. Tensors are generally high-dimensional arrays that generalize vectors and matrices. If all dimensions except one of a d-way tensor are equal to 1 (i.e. $I_k = 1$, for $k = 1, \ldots, t-1, t+1, \ldots, d$), the d-order tensor is essentially a vector of length I_t. Alternatively, any vector or matrix can be viewed as a d-way tensor by appending augmented dimensions that are size 1. We use the convention $1 \leq i_k \leq I_k$ for $k = 1, \ldots, d$. MATLAB notation is used to denote entries of tensors. In this paper, boldface capital calligraphic letters $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \ldots$ are used to denote tensors, boldface capital letters $\boldsymbol{A}, \boldsymbol{B}, \ldots$ denote matrices, boldface letters $\boldsymbol{a}, \boldsymbol{b}, \ldots$ denote vectors, and Roman letters $a, b, \ldots$ denote scalars. A set of d tensors, like that of a tensor train, is denoted as $\boldsymbol{\mathcal{A}}^{(1)}, \boldsymbol{\mathcal{A}}^{(2)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}$. A useful representation of tensors are tensor network diagrams. These diagrams use a graphical depiction of scalars, vectors, matrices, and tensors as introduced in Figure 1, where each node represents a tensor and the number of free edges on each node represents its order. For example, matrices are 2-way tensors, and thus are represented by nodes with two unconnected edges. We now give a brief description of some required tensor operations. The generalization of the matrix-matrix multiplication to tensors involves a multiplication of a matrix with a d-way tensor along one of its d modes.

Definition 1: ([8, p. 460]) The k-mode product of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with a matrix $\boldsymbol{U} \in \mathbb{R}^{J \times I_k}$ is denoted $\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{A}} \times_k \boldsymbol{U}$ and defined by

$$\boldsymbol{\mathcal{B}}(i_1, \cdots, i_{k-1}, j, i_{k+1}, \cdots, i_d) = \sum_{i_k=1}^{I_k} \boldsymbol{U}(j, i_k) \boldsymbol{\mathcal{A}}(i_1, \cdots, i_{k-1}, i_k, i_{k+1}, \cdots, i_d),$$

where $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times J \times I_{k+1} \times \cdots \times I_d}$.

The proposed method also requires the knowledge of the matrix Khatri-Rao product, which is crucial in the construction of the input matrix $\boldsymbol{S}$.

Definition 2: If $\boldsymbol{A} \in \mathbb{R}^{N_1 \times M}$ and $\boldsymbol{C} \in \mathbb{R}^{N_2 \times M}$, then their Khatri-Rao product $\boldsymbol{A} \odot \boldsymbol{C}$ is the $N_1 N_2 \times M$ matrix

$$\left(\boldsymbol{A}(:, 1) \otimes \boldsymbol{C}(:, 1) \; \cdots \; \boldsymbol{A}(:, M) \otimes \boldsymbol{C}(:, M)\right),$$

where $\otimes$ denotes the standard Kronecker product.

Two other basic operations include tensor reshaping and vectorization.

Definition 3: We adopt the MATLAB reshape operator "reshape($\boldsymbol{\mathcal{A}}$, $[n_1, n_2, n_3 \cdots]$)", which reshapes the d-way tensor $\boldsymbol{\mathcal{A}}$ with column-wise ordering preserved into a tensor with dimensions $n_1 \times n_2 \times \cdots \times n_d$. The total number of elements of $\boldsymbol{\mathcal{A}}$ must be the same as $n_1 \times n_2 \times \cdots \times n_d$.

Definition 4: The vectorization vec($\boldsymbol{\mathcal{A}}$) of a tensor $\boldsymbol{\mathcal{A}}$ is the vector obtained from concatenating all tensor entries into one column vector.

More details on these tensor operations can be found in [8, p. 459]. The mapping between the index i of the vector vec($\boldsymbol{\mathcal{A}}$) and the multi-index $[i_1, i_2, \ldots, i_d]$ of the corresponding tensor $\boldsymbol{\mathcal{A}}$ is bijective. Herein we adopt the mapping convention

$$i = i_1 + \sum_{k=2}^{d} (i_k - 1) \prod_{p=1}^{k-1} I_p \tag{1}$$

to convert the multi-index $[i_1, i_2, \ldots, i_d]$ into the linear index i. Before going into the formal definition of a tensor train decomposition, we give an intuitive interpretation of this process in Figure 2. For a given 4-way tensor as the one on the left-hand side, we analogize it to a large train cabin with four wheels (legs), while a tensor train decomposition separates the train cabin to four small connected cabins as shown on the right-hand side. Using a tensor train format can reduce the storage complexity of a tensor significantly. The storage of a tensor train of a d-way tensor $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I \times I \times \cdots \times I}$ requires approximately dIR^2 elements, where R is the maximum TT-rank, compared to the conventional I^d storage requirement. The formal definition is given as follows:

Definition 5: A tensor train of a vector $\boldsymbol{a} \in \mathbb{R}^{I_1 I_2 \cdots I_d}$ is defined as the set of d 3-way tensors $\boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}$ such that $\boldsymbol{a}(i)$ equals

$$\sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \boldsymbol{\mathcal{A}}^{(1)}(r_1, i_1, r_2) \, \boldsymbol{\mathcal{A}}^{(2)}(r_2, i_2, r_3) \cdots \boldsymbol{\mathcal{A}}^{(d)}(r_d, i_d, r_1), \tag{2}$$

where i is related to $[i_1, i_2, \ldots, i_d]$ via (1) and $r_1, r_2, \ldots, r_d$ are auxiliary indices that are summed over. The dimensions $R_1, R_2, \ldots, R_d$ of these auxiliary indices are called the tensor train ranks (TT-ranks).

The summations over the auxiliary indices are represented in Figure 2 by the connected edges between the different nodes. The second auxiliary index of $\boldsymbol{\mathcal{A}}^{(d)}$ is R_1, which ensures that the summation in (2) results in a scalar. When $R_1 > 1$, the tensor train is also called a tensor ring or matrix product state with periodic boundary conditions [33], [34]. Throughout this article we always choose $R_1 = 1$.

The notion of a site-k-mixed canonical tensor train is very useful when implementing our proposed algorithm. In order to be able to define this notion, we first need to introduce left and right-orthogonal tensor train cores.

Definition 6: ([31, p. A689]) A tensor train core $\boldsymbol{\mathcal{A}}^{(k)}$ is left-orthogonal if it can be reshaped into an $R_k I_k \times R_{k+1}$ matrix $\boldsymbol{A}$ for which

$$\boldsymbol{A}^T \boldsymbol{A} = \boldsymbol{I}_{R_{k+1}}$$

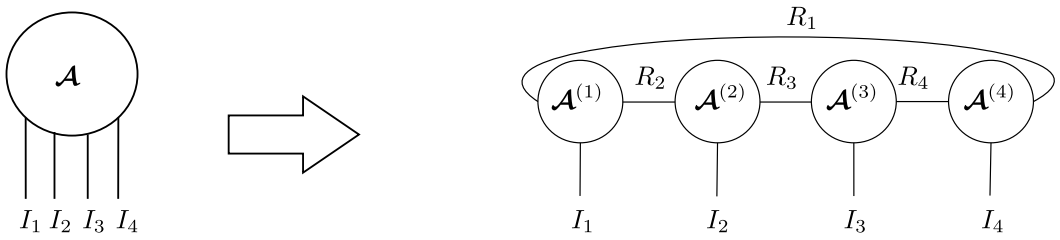


Fig. 2. Graphical depiction of the tensor train decomposition of a 4-way tensor $\mathcal{A}$.

applies. Similarly, a tensor train core $\mathcal{A}^{(k)}$ is right-orthogonal if it can be reshaped into an $R_k \times I_k R_{k+1}$ matrix $\boldsymbol{A}$ for which

$$\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}_{R_k}$$

applies.

Definition 7: (Site-k-mixed-canonical tensor train) A tensor train is in site-k-mixed-canonical form [35] when all TT-cores $\mathcal{A}^{(l)}$ $(1 \leq l \leq k-1)$ are left-orthogonal and TT-cores $\mathcal{A}^{(l)}$ $(k+1 \leq l \leq d)$ are right-orthogonal.

One advantage of a tensor $\mathcal{A}$ being in a site-k-mixed-canonical form is that its Frobenius norm is easily computed as

$$||\mathcal{A}||_F^2 = ||\mathcal{A}^{(k)}||_F^2 = \text{vec}(\mathcal{A}^{(k)})^T \text{vec}(\mathcal{A}^{(k)}). \tag{3}$$

III. Methodology

We first explain the basic idea of our proposed method without any TV or Tikhonov regularization in Sections III-A up to III-D. Discussions on how the tensor dimensions can be factorized and the TT-ranks chosen are given in Section III-E. The inclusion of both TV and Tikhonov regularization are discussed in Sections III-F and III-G, respectively.

A. Basic Idea

The proposed tensor completion method intrinsically relies on solving an underdetermined linear system under a low TT-rank constraint. For a given set of N multi-indices $[i_1, i_2, \ldots, i_d]$ and a corresponding vector of observed tensor entries $\boldsymbol{y} \in \mathbb{R}^N$, the goal is to obtain a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ that contains the same tensor entries. Equivalently, we form the optimization problem

$$\begin{aligned} &\min_{\mathcal{A} \in \mathcal{S}_{\text{TT}}^{(d)}} ||\boldsymbol{S}^T \text{ vec}(\mathcal{A}) - \boldsymbol{y}||_2^2, \\ &\text{s.t. } \mathcal{A} \in \mathcal{S}_{\text{TT}}^{(d)}, \quad \text{and TT-rank}(\mathcal{A}) = (R_1, R_2, \ldots, R_d), \end{aligned} \tag{4}$$

where $\boldsymbol{S}^T \in \mathbb{R}^{N \times I_1 I_2 I_3 \cdots I_d}$ is the row selection matrix corresponding with the known multi-indices. In other words, we want $\boldsymbol{S}^T \text{vec}(\mathcal{A}) \in \mathbb{R}^{N \times 1}$ to be as close as possible to the observed entries $\boldsymbol{y}$, under the constraint that vec$(\mathcal{A})$ has a low-rank tensor train representation. We stress that the tensor $\mathcal{A}$ is never computed explicitly in the proposed algorithm but is stored in its tensor train format instead. This is possible by exploiting the fact that $\boldsymbol{S}$ can be written as the Khatri-Rao product of d smaller binary matrices

$$\boldsymbol{S} = \boldsymbol{S}^{(d)} \odot \boldsymbol{S}^{(d-1)} \odot \ldots \odot \boldsymbol{S}^{(1)}. \tag{5}$$

Equation (5) also implies that the matrix $\boldsymbol{S}$ never needs to be explicitly kept in memory. By storing the factor matrices $\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(d)}$ instead, the storage cost for $\boldsymbol{S}$ is reduced from $N(I_1 \cdots I_d)$ down to $N(I_1 + \cdots + I_d)$. The decomposition of $\boldsymbol{S}$ follows from the following definition.

Definition 8: For a tensor entry $\mathcal{A}(i_1, i_2, \ldots, i_d)$, the corresponding selection vector $\boldsymbol{s}_{[i_1, i_2, \ldots, i_d]}$ is defined as

$$\boldsymbol{s}_{[i_1, i_2, \ldots, i_d]} := \boldsymbol{e}_{\boldsymbol{i_d}}^{(d)} \otimes \cdots \otimes \boldsymbol{e}_{\boldsymbol{i_2}}^{(2)} \otimes \boldsymbol{e}_{\boldsymbol{i_1}}^{(1)}, \tag{6}$$

where $\boldsymbol{e}_{\boldsymbol{i_k}}^{(k)} \in \mathbb{R}^{I_k}$ $(k = 1, 2, \ldots, d)$ is the i_k-th standard basis vector.

One can verify that

$$\boldsymbol{s}_{[i_1, i_2, \ldots, i_d]}^T \text{vec}(\mathcal{A}) = \mathcal{A}(i_1, i_2, \ldots, i_d).$$

Note that the order of the Kronecker products is reversed to be consistent with the index mapping (1). We now define the n-th column of the $N \times I_k$ selection matrix $\boldsymbol{S}^{(k)}$ as the standard basis vector $\boldsymbol{e}_{\boldsymbol{i_k}}^{(k)}$ of the n-th observed entry. Equation (5) then follows from the concatenation of (6) for each known multi-index.

Example 1: We use a small example to illustrate Definition 8. Consider a $3 \times 4 \times 2$ tensor $\mathcal{A}$ with only 3 of the entries observed. The multi-indices of the three observed entries are

$$[2, 1, 2], [1, 3, 1], [3, 4, 2].$$

The corresponding selection vectors are then given by

$$\begin{aligned} \boldsymbol{s}_{[2,1,2]} &= \boldsymbol{e}_{\boldsymbol{2}}^{(3)} \otimes \boldsymbol{e}_{\boldsymbol{1}}^{(2)} \otimes \boldsymbol{e}_{\boldsymbol{2}}^{(1)}, \\ \boldsymbol{s}_{[1,3,1]} &= \boldsymbol{e}_{\boldsymbol{1}}^{(3)} \otimes \boldsymbol{e}_{\boldsymbol{3}}^{(2)} \otimes \boldsymbol{e}_{\boldsymbol{1}}^{(1)}, \\ \boldsymbol{s}_{[3,4,2]} &= \boldsymbol{e}_{\boldsymbol{2}}^{(3)} \otimes \boldsymbol{e}_{\boldsymbol{4}}^{(2)} \otimes \boldsymbol{e}_{\boldsymbol{3}}^{(1)}. \end{aligned}$$

The corresponding selection matrices for each mode are then

$$\begin{aligned} \boldsymbol{S}^{(1)} &= \left(\boldsymbol{e}_{\boldsymbol{2}}^{(1)} \; \boldsymbol{e}_{\boldsymbol{1}}^{(1)} \; \boldsymbol{e}_{\boldsymbol{3}}^{(1)}\right) \in \mathbb{R}^{3 \times 3}, \\ \boldsymbol{S}^{(2)} &= \left(\boldsymbol{e}_{\boldsymbol{1}}^{(2)} \; \boldsymbol{e}_{\boldsymbol{3}}^{(2)} \; \boldsymbol{e}_{\boldsymbol{4}}^{(2)}\right) \in \mathbb{R}^{4 \times 3}, \\ \boldsymbol{S}^{(3)} &= \left(\boldsymbol{e}_{\boldsymbol{2}}^{(3)} \; \boldsymbol{e}_{\boldsymbol{1}}^{(3)} \; \boldsymbol{e}_{\boldsymbol{2}}^{3)}\right) \in \mathbb{R}^{2 \times 3}. \end{aligned}$$

In our regression task, the d matrices $\boldsymbol{S}^{(1)}, \boldsymbol{S}^{(2)}, \ldots, \boldsymbol{S}^{(d)}$ from (5) act as the inputs to the unknown model and the output is the vector of N observed tensor entries

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_N)^T,$$

where typically $N \ll I_1 I_2 \cdots I_d$.

B. Tensor Train Initialization

The proposed tensor completion algorithm solves (4) iteratively using an alternating linear scheme (ALS). Starting from an initial guess for the tensor train of $\text{vec}(\mathcal{A})$, the ALS updates each tensor train core for a predefined number of iterations or until convergence. Each tensor train core update is achieved by solving a relatively small (compared to the original tensor size) least squares problem. A good initial guess is therefore of crucial importance to speed up convergence. Through extensive tests, we observed that the following heuristical initialization method for images and videos is effective in terms of convergence speed.

Suppose $\mathcal{V} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is a tensor with missing entries. Before converting $\mathcal{V}$ into a tensor train, the goal is to fill up the missing entries using information from the observed entries through two interpolation steps. First, each dimension of the tensor $\mathcal{V}$ is resized by a factor h using a box-shaped interpolation kernel. The resulting tensor is denoted $\mathcal{W} \in \mathbb{R}^{\lfloor \frac{I_1}{h} \rfloor \times \lfloor \frac{I_2}{h} \rfloor \times \cdots \times \lfloor \frac{I_d}{h} \rfloor}$ and its entries are then used to construct a tensor of the original size $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ through cubic kernel interpolation. Alternatively, one can use max or average pooling together with interpolation to achieve a similar effect. Note that for color images and videos the color dimension is not resized during the whole initialization procedure. We refer the readers to Section IV-A.5 for comparing the convergence behavior of the proposal versus a vanilla zero initialization. Finally, a tensor train with given TT-ranks is computed from $\mathcal{X}$ by a modified version of the TT-SVD algorithm [30, p. 2301]. The TT-SVD algorithm decomposes a tensor to its tensor train format by consecutive reshapings and singular value decomposition (SVD). The modification is made in line 5 of the TT-SVD algorithm, where instead of using the original truncation parameter δ, each SVD is truncated to the prescribed TT-rank $R_2, \ldots, R_d$. Alternatively, one can use a Krylov subspace method ('svds' command in MATLAB) to determine the desired truncated SVD. Using the TT-SVD algorithm to obtain the initial estimate of tensor train also implies that the tensor train will be in site-d-mixed-canonical form. We provide a sketch of the proof:

Proof: A typical TT-SVD starts from the truncated SVD, $\boldsymbol{U}_L^{(1)} \boldsymbol{\Sigma}^{(1)} \boldsymbol{U}_R^{(1)T} = \text{reshape}(\mathcal{A}, [I_1, I_2 \ldots I_d])$. Since the entries of $\mathcal{A}$ are real numbers, we have $\boldsymbol{U}_L^{(1)T} \boldsymbol{U}_L^{(1)} = \boldsymbol{I}_{R_2}, \boldsymbol{\Sigma} \boldsymbol{U}_R^{(1)T} \in \mathbb{R}^{R_2 \times I_2 \ldots I_d}$. The first TT-core is then constructed as $\text{reshape}(\boldsymbol{U}_L^{(1)}, [1, I_2, R_2])$, which is automatically left-orthogonal by definition. The norm $||\mathcal{A}||_F$ therefore resides in the $\boldsymbol{\Sigma} \boldsymbol{U}_R^{(1)T}$ part. Next, a truncated SVD is operated on $\text{reshape}(\boldsymbol{\Sigma} \boldsymbol{U}_R^{(1)T}, [R_2 I_2, I_3 \ldots I_d]) = \boldsymbol{U}_L^{(2)} \boldsymbol{\Sigma}^{(2)} \boldsymbol{U}_R^{(2)T}$, and the second TT-core is formed by $\text{reshape}(\boldsymbol{U}_L^{(2)}, [R_2, I_2, R_3])$ that naturally satisfies left-orthogonalty as well. At this point we have two left orthogonal TT-cores and a tensor that contains the norm $||\mathcal{A}||_F$. By continued application of this reshaping and a truncated SVD we subsequently obtain a site-d-mixed-canonical initial tensor train. ∎

C. Alternating Linear Scheme

We now derive the least squares problem for updating each tensor train core during the ALS. The main motivation for solving (4) in tensor train form is the reduction in computational cost. Indeed, we will show how updating each tensor train core $\mathcal{A}^{(k)}$ during the ALS has a computational cost of $O(N(R_k I_k R_{k+1})^2)$ flops, whereas in vector form the computational cost would take approximately $O(N^2 I^d)$ flops. In addition, by specifying small TT-ranks one effectively regularizes the problem since the underdetermined system $\boldsymbol{S}^T \text{vec}(\mathcal{A}) = \boldsymbol{y}$ will typically have an infinite number of solutions. In what follows, $\boldsymbol{s}_l^{(k)} \in \mathbb{R}^{I_k \times 1} (1 \leq l \leq N)$ denotes the l-th column of the matrix $\boldsymbol{S}^{(k)}$. We further define the following useful auxiliary notations

$$\boldsymbol{a}_{<k,l}^T := (\mathcal{A}^{(1)} \times_2 \boldsymbol{s}_l^{(1)T}) \ldots (\mathcal{A}^{(k-1)} \times_2 \boldsymbol{s}_l^{(k-1)T}) \in \mathbb{R}^{R_k},$$
$$\boldsymbol{a}_{>k,l} := (\mathcal{A}^{(k+1)} \times_2 \boldsymbol{s}_l^{(k+1)T}) \ldots (\mathcal{A}^{(d)} \times_2 \boldsymbol{s}_l^{(d)T}) \in \mathbb{R}^{R_{k+1}},$$

for $k = 2, \ldots, d-1$. Per definition $\boldsymbol{a}_{<1,l} = \boldsymbol{a}_{>d,l} = 1$. The l-th observed entry $\boldsymbol{y}(l)$ can then be written as

$$\boldsymbol{y}(l) = (\boldsymbol{a}_{>k,l}^T \otimes \boldsymbol{s}_l^{(k)T} \otimes \boldsymbol{a}_{<k,l}) \, \text{vec}(\mathcal{A}^{(k)}). \tag{7}$$

The proof of equation (7) resembles that in [36, Theorem 4.1]. Writing out (7) for all N observed entries results in the following linear system

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{a}_{>k,1}^T \otimes \boldsymbol{s}_1^{(k)T} \otimes \boldsymbol{a}_{<k,1} \\ \boldsymbol{a}_{>k,2}^T \otimes \boldsymbol{s}_2^{(k)T} \otimes \boldsymbol{a}_{<k,2} \\ \vdots \\ \boldsymbol{a}_{>k,N}^T \otimes \boldsymbol{s}_N^{(k)T} \otimes \boldsymbol{a}_{<k,N} \end{pmatrix} \text{vec}(\mathcal{A}^{(k)}), \tag{8}$$

where the matrix is $N \times R_k I_k R_{k+1}$. Solving (8) requires $O(N(R_k I_k R_{k+1})^2)$ flops. If the TT-ranks R_k, R_{k+1} are chosen such that $N \geq R_k I_k R_{k+1}$, one can solve the normal equations of (8) instead with a computational complexity of $O((R_k I_k R_{k+1})^3)$ flops, which comes at the cost of a squared condition number and possible loss of accuracy. It is possible to construct the matrix of (8) without computing any Kronecker product by exploiting the structure of the binary $\boldsymbol{S}^{(k)}$ matrices. However, the total runtime of our proposed algorithm is dominated by solving the linear system (8) so we will not discuss this particular optimization any further. The key idea of the ALS is to solve (8) for varying values of k in a "sweeping" fashion, starting from the leftmost tensor train core $\mathcal{A}^{(1)}$ to the rightmost $\mathcal{A}^{(d)}$ and then back from the rightmost to the leftmost. The numerical stability of the ALS algorithm is guaranteed through an orthogonalization step using the QR decomposition [31, p. A701].

D. Tensor Train Completion Algorithm

The pseudocode of the proposed tensor train completion (TTC) algorithm is given as Algorithm 1. First, d binary input matrices $\boldsymbol{S}^{(1)}, \boldsymbol{S}^{(2)}, \ldots, \boldsymbol{S}^{(d)}$ are constructed as specified in Section III. The tensor train with specified TT-ranks is then initialized according to Section III-B. The ALS is then applied to update the tensor train cores one by one in a sweeping fashion by solving (8) repeatedly for different values of k. Since the tensor train is in site-d-mixed-canonical form (see proof III-B in Section IV-A.5 for details), the updating starts with $\mathcal{A}^{(d)}$. One can update the tensor train cores for a fixed amount of sweeps or until the residual falls below a certain

Algorithm 1 Tensor Completion in Tensor Train Form (TTC)

Input: d-way multi-indices of N known entries and their corresponding values $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(N)$, TT ranks $R_2, \ldots, R_d$.
Output: Completed tensor $\boldsymbol{\mathcal{A}}$ in tensor train form $\boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}$.

1: Construct $\boldsymbol{S}^{(1)}$,$\boldsymbol{S}^{(2)}$,...,$\boldsymbol{S}^{(d)}$ as specified in Section III-A.
2: Initialize the tensor train as specified in Section III-B.
3: **while** stopping criteria not satisfied **do**
4: **for** k=d,...,2 **do**
5: vec($\boldsymbol{\mathcal{A}}^{(k)}$) $\leftarrow$ solve equation (8), (11) or (13)
6: $\boldsymbol{A}_k \leftarrow$ reshape($\boldsymbol{\mathcal{A}}^{(k)}, [R_k, I_k R_{k+1}]$)
7: $[\boldsymbol{Q}, \boldsymbol{R}] \leftarrow$ thin QR decomposition of $\boldsymbol{A}_k^T$
8: $\boldsymbol{\mathcal{A}}^{(k)} \leftarrow$ reshape($\boldsymbol{Q}^T, [R_k, I_k, R_{k+1}]$)
9: $\boldsymbol{A}_{k-1} \leftarrow$ reshape($\boldsymbol{\mathcal{A}}^{(k-1)}, [R_{k-1} I_{k-1}, R_k]$)
10: $\boldsymbol{\mathcal{A}}^{(k-1)} \leftarrow$ reshape($\boldsymbol{A}_{k-1} \boldsymbol{R}^T, [R_{k-1}, I_{k-1}, R_k]$)
11: **end for**
12: **for** $k = 1, \ldots, d-1$ **do**
13: vec($\boldsymbol{\mathcal{A}}^{(k)}$) $\leftarrow$ solve equation (8), (11) or (13)
14: $\boldsymbol{A}_k \leftarrow$ reshape($\boldsymbol{\mathcal{A}}^{(k)}, [R_k I_k, R_{k+1}]$)
15: $[\boldsymbol{Q}, \boldsymbol{R}] \leftarrow$ thin QR decomposition of $\boldsymbol{A}_k$
16: $\boldsymbol{\mathcal{A}}^{(k)} \leftarrow$ reshape($\boldsymbol{Q}, [R_k, I_k, R_{k+1}]$)
17: $\boldsymbol{A}_{k+1} \leftarrow$ reshape($\boldsymbol{\mathcal{A}}^{(k+1)}, [R_{k+1}, I_{k+1} R_{k+2}]$)
18: $\boldsymbol{\mathcal{A}}^{(k+1)} \leftarrow$ reshape($\boldsymbol{R} \boldsymbol{A}_{k+1}, [R_{k+1}, I_{k+1}, R_{K+2}]$)
19: **end for**
20: **end while**

threshold. Numerical stability and convergence is guaranteed by the QR factorization step in line 7. In lines 8 to 10 of the algorithm the updated tensor $\boldsymbol{\mathcal{A}}^{(k)}$ is replaced by a reshaping of the orthogonal $\boldsymbol{Q}$ matrix and the $\boldsymbol{R}$ factor is "absorbed" into the next core $\boldsymbol{\mathcal{A}}^{(k-1)}$. In this way, the resulting tensor train is brought into site-$(k-1)$-mixed-canonical form, before updating $\boldsymbol{\mathcal{A}}^{(k-1)}$. Global convergence to the solution with unique minimal norm is not guaranteed. Once $\boldsymbol{\mathcal{A}}^{(2)}$ has been updated, one iteration has completed and the sweep reverses direction with updating the tensor train cores from $\boldsymbol{\mathcal{A}}^{(1)}$ up to $\boldsymbol{\mathcal{A}}^{(d-1)}$. Each for-loop in Algorithm 1 therefore corresponds with one iteration. Depending on the application one can either choose to keep the result in tensor train form or compute the full tensor $\boldsymbol{\mathcal{A}}$ by summing the tensor train over its auxiliary indices.

The most computationally expensive steps in Algorithm 1 are solving the linear systems in line 5 and line 13.

E. Choosing a Dimension Factorization and TT-Ranks

Color images and videos are 3-way and 4-way tensors, respectively. Converting these tensors directly into tensor trains would then result in 3 or 4 TT-cores with relatively large I_k dimensions. This potentially has a detrimental effect on the runtime of Algorithm 1 as the computational complexity of solving (8) is $O((R_k I_k R_{k+1})^3)$. The runtime of computing inverse matrices of different sizes varies from one computer to another (the coefficients of the cubic complexity function are dependent on the computer specifications). For the desktop computer used in our experiments, a sharp increase in runtime for matrix inversion was observed when $R_k I_k R_{k+1} \approx 4000$. The problem size was therefore limited to $R_k I_k R_{k+1} \leq 4000$ in our experiments to guarantee a fast completion. This problem size limitation was implemented by factorizing each of the I_k dimensions of the desired completed tensor into more manageable sizes. This implies that each linear index of I_k is split into a corresponding multi-index. This factorization comes at the cost of introducing more TT-cores, resulting in a trade-off between the total number of TT-cores and the total number of parameters per TT-core. The maximal amount of TT-cores is determined by the factorization of each tensor dimension I_k into its prime factors. Specifically, we choose the dimensions of each tensor train core $I_k \leq 10$ combined with TT-ranks $R_k R_{k+1} \leq 400$ for $k = 1, \ldots, d$. We will show in an experiment that as long as we follow the above rules, the completion performance of different factorization sizes are similar. Moreover, in case the dimensions are large prime numbers, one can always append zeros to the tensor such that the dimensions can be factored. Since we never mix column indices and row indices, the tensor decomposition essentially serves as a normal matrix SVD with the left and right singular vectors being factorized intro Kronecker products of smaller vectors with low-rank constraints.

Example 2: Suppose we have a $360 \times 640 \times 144 \times 3$ tensor of a color video that consists of 144 frames. The prime factorizations of 360, 640 and 144 are $2^3 \times 3^2 \times 5$, $2^7 \times 5$ and $2^4 \times 3^2$, respectively. Separating each of these factors into TT-cores would result in a tensor train of 21 cores. The number of cores can, for example, be set to 11 by using the factorization $9 \times 8 \times 5 \times 4 \times 4 \times 5 \times 8 \times 4 \times 6 \times 6 \times 3$.

Choosing good values for the $d-1$ TT-ranks can be quite tedious when d is large. We therefore propose to choose the values of R_2, R_{mid} ("mid" stands for middle) and R_d and automatically determine the remaining TT-ranks $R_{k+1} (2 \leq k \leq d-2)$ as $\min(R_k I_k, R_{\text{mid}})$. In this way, a uniform plateau of TT-ranks equal to R_{mid} is obtained. The TT-ranks R_2, R_d need to be chosen while keeping in mind that $R_2 \leq I_1$ and $R_d \leq I_d$. When the tensor represents either a color image or color video, the last dimension will typically be 3 for the color channels. In this case, we always set $R_d = 3$ and additionally choose the value of R_{d-1}.

Example 3: We revisit the $360 \times 640 \times 144 \times 3$ tensor of Example 2, together with the 11-core dimension factorization $9 \times 8 \times 5 \times 4 \times 4 \times 5 \times 8 \times 4 \times 6 \times 6 \times 3$. Choosing $R_2 = 5$, $R_{\text{mid}} = 5$, $R_{d-1} = R_{10} = 5$ and $R_{11} = 3$ then results in $R_3 = \cdots = R_9 = 5$.

F. Total Variation Regularization

The low TT-rank constraint in (4) can be interpreted as a global feature, as it pertains to the construction of the whole tensor. Better completion results can be obtained from the addition of local constraints on the completed tensor entries. The notion of local smoothness, as described by TV, is such a local feature, and is particularly useful when the tensor represents visual data. The addition of TV terms to (4) is quite straightforward, resulting in

$$\min_{\boldsymbol{\mathcal{A}} \in \mathcal{S}_{\text{TT}}^{(d)}} ||\boldsymbol{S}^T \text{ vec}(\boldsymbol{\mathcal{A}}) - \boldsymbol{y}||_2^2 + \sum_{p=1}^{P} \lambda_p \, ||\boldsymbol{\mathcal{A}} \times_p \boldsymbol{D}_p||_F^2,$$
$$\text{s.t. } \boldsymbol{\mathcal{A}} \in \mathcal{S}_{\text{TT}}^{(d)}, \quad \text{and TT-rank}(\boldsymbol{\mathcal{A}}) = (R_1, R_2, \ldots, R_d), \quad (9)$$

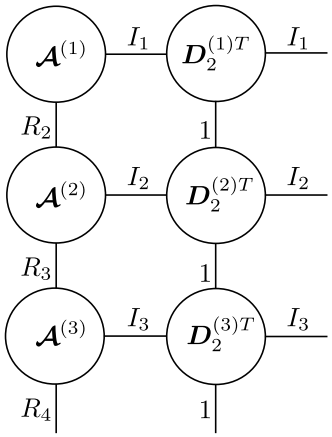


Fig. 3. Tensor network diagram of $\boldsymbol{W}_{<4,2}$ with the $\boldsymbol{D}_2^{(1)}$, $\boldsymbol{D}_2^{(2)}$, and $\boldsymbol{D}_2^{(3)}$ being $\boldsymbol{I}_{I_1}$, $\boldsymbol{D}_2$, and $\boldsymbol{I}_{I_3}$, respectively.

where $\boldsymbol{D}_p \in \mathbb{R}^{(I_p-1)\times I_p}$ has entries $\boldsymbol{D}_p(i,i) = 1$ and $\boldsymbol{D}_p(i,i+1) = -1$. For notational convenience we make $\boldsymbol{D}_p$ square by appending a bottom row of zeros. Minimizing each of the $||\boldsymbol{\mathcal{A}} \times_p \boldsymbol{D}_p||_F^2$ terms ensures that the completed tensor entries do not differ too much from their neighbors along modes $1, \ldots, P$, which encodes local smoothness. For image and video tensors we have $P = 2$, as the smoothness is only required in the width and length dimensions. Solving (9) requires the modification of (8) with additional matrix terms. To derive how exactly (8) needs to be modified, we will first ignore the fact that we can factorize the dimensions of the original tensor as discussed in Section III-E. It is worthwhile to stress that to the best of our knowledge, this is the first time that the Total Variation regularization term is exploited and fully incorporated in tensor train form.

The derivation of this modification is very similar to the traditional ALS derivation, where the input matrices $\boldsymbol{S}^{(k)}$ are now replaced with the matrices

$$\boldsymbol{D}_p^{(k)} := \begin{cases} \boldsymbol{D}_p, & \text{if } k = p, \\ \boldsymbol{I}_{I_k}, & \text{otherwise.} \end{cases} \tag{10}$$

Indeed, the p-th TV regularization term can now be rewritten as $(\boldsymbol{D}_p^{(d)} \otimes \cdots \otimes \boldsymbol{D}_p^{(2)} \otimes \boldsymbol{D}_p^{(1)})\,\text{vec}(\boldsymbol{\mathcal{A}})$. The only difference between the TV term and $\boldsymbol{S}^T\,\text{vec}(\boldsymbol{\mathcal{A}})$ is that $\boldsymbol{S}$ consists of a Khatri-Rao product, while the TV term contains a Kronecker product. Just as in Section III-D we consider the contractions of the TT-cores of $\boldsymbol{\mathcal{A}}$ with these new "input" matrices

$$\boldsymbol{W}_{<k,p} := (\boldsymbol{\mathcal{A}}^{(1)} \times_2 \boldsymbol{D}_p^{(1)}) \ldots (\boldsymbol{\mathcal{A}}^{(k-1)} \times_2 \boldsymbol{D}_p^{(k-1)}),$$
$$\boldsymbol{W}_{>k,p} := (\boldsymbol{\mathcal{A}}^{(k+1)} \times_2 \boldsymbol{D}_p^{(k+1)}) \ldots (\boldsymbol{\mathcal{A}}^{(d)} \times_2 \boldsymbol{D}_p^{(d)}),$$

which allows us to define the matrix

$$\boldsymbol{W}_p := (\boldsymbol{W}_{>k,p} \otimes \boldsymbol{D}_p^{(k)} \otimes \boldsymbol{W}_{<k,p}) \in \mathbb{R}^{\prod_{i=1}^{d} I_i \times R_k I_k R_{k+1}}.$$

In Figure 3, we exemplify the calculation of $\boldsymbol{W}_{<k,p}$ using tensor diagrams. Specifically, we give an example of $\boldsymbol{W}_{<4,2}$ ($k = 4$, $p = 2$), where, according to Equation (10), we have $\boldsymbol{D}_2^{(1)} = \boldsymbol{I}_{I_1}$, $\boldsymbol{D}_2^{(2)} = \boldsymbol{D}_2$, and $\boldsymbol{D}_2^{(3)} = \boldsymbol{I}_{I_3}$. Denoting the matrix in (8) by $\boldsymbol{B}$, the modified linear system is then

$$(\boldsymbol{B}^T\boldsymbol{B} + \sum_{p=1}^{P} \lambda_p \boldsymbol{W}_p^T \boldsymbol{W}_p)\,\text{vec}(\boldsymbol{\mathcal{A}}^{(k)}) = \boldsymbol{B}^T\,\boldsymbol{y}. \tag{11}$$

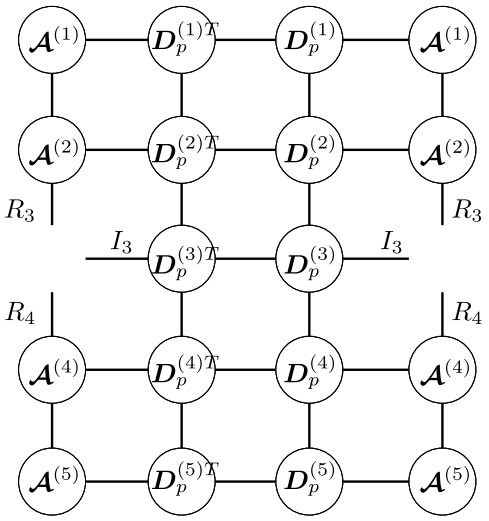


Fig. 4. Tensor network diagram of $\boldsymbol{W}_p^T\boldsymbol{W}_p$ when updating $\boldsymbol{\mathcal{A}}^{(3)}$ in a tensor train of 5 TT-cores.

The number of rows of $\boldsymbol{W}_p$ grows exponentially with d. Fortunately, it is not necessary to explicitly construct this matrix as it is possible to compute $\boldsymbol{W}_p^T\boldsymbol{W}_p$ directly from the tensor network diagram depicted in Figure 4 for the case $d = 5, k = 3$. This computation can be done efficiently by exploiting the fact that most of the $\boldsymbol{D}_p^{(i)}$ matrices are the unit matrix. In addition, the site-k-mixed-canonical form of the tensor train of $\boldsymbol{\mathcal{A}}$ can also be exploited. Suppose for example that $p = 1$ in Figure 4. The site-3-mixed-canonical form of the tensor train of $\boldsymbol{\mathcal{A}}$ then implies that both $\boldsymbol{\mathcal{A}}^{(4)}$ and $\boldsymbol{\mathcal{A}}^{(5)}$ are right-orthogonal. Since $p = 1$, both $\boldsymbol{D}_1^{(4)}$ and $\boldsymbol{D}_1^{(5)}$ matrices are unit matrices. The summation of the two bottom rows in Figure 4 then results in an $R_4 \times R_4$ unit matrix and can be skipped. Also note that all vertical edges between the different $\boldsymbol{D}_p^{(k)}$ matrices have dimensions 1.

When the dimensions of the original tensor are factorized as discussed in Section III-E, then only one minor modification is required. The TT-SVD algorithm needs to be applied to $\boldsymbol{D}_p$ to transform this matrix into a tensor train matrix (TT matrix) [37] according to the same dimension factorization. The corresponding node in the tensor network diagram is then replaced by the corresponding TT matrix. As a result, not all vertical edges will have dimension 1 anymore. It turns out that the TT matrix-ranks of $\boldsymbol{D}_p$ are always uniformly 3, irrespective of the factorization of the dimensions or number of cores. This low TT-rank feature of the $\boldsymbol{D}_p$ matrix is favorable for the computation of $\boldsymbol{W}_p^T\boldsymbol{W}_p$ from the tensor network.

Example 4: Suppose we have a $1024 \times 1024 \times 3$ color image and we factor each of the 1024 dimensions into 4^5. The tensor train of $\text{vec}(\boldsymbol{\mathcal{A}})$ hence consists of 11 cores. The 1024×1024 $\boldsymbol{D}_p$ matrix is then converted into a TT matrix of 5 cores with ranks $R_2 = \cdots = R_5 = 3$. When computing the $\boldsymbol{W}_1^T\boldsymbol{W}_1$ term, we then have that the first five TT-cores of $\boldsymbol{\mathcal{A}}$ are connected with the TT matrix of $\boldsymbol{D}_1$, while the remaining cores are connected to identity matrices.

The values of the λ_p parameters were fixed to 0.5 in [18]. Through experiments, we found that choosing values between 0 and 10 for the λ_p parameters resulted in consistently

TABLE I
PERFORMANCE (RSES) OF TEN ALGORITHMS ON EIGHT BENCHMARK IMAGES

Method	House		River		Bridge		Man		Lena		Peppers		Baboon		Airplane	
	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)
TNN	0.221	44.9	0.193	43.0	0.159	60.7	0.231	33.5	0.221	13.6	0.295	16.6	0.246	15.7	0.168	18.1
HaLRTC	0.209	7.9	0.182	6.3	0.153	15.6	0.221	20.3	0.206	10.6	0.265	5.2	0.222	4.1	0.162	8.4
FaLRTC	0.210	30.0	0.181	10.0	0.153	26.6	0.218	10.6	0.202	5.1	0.253	5.3	0.228	7.0	0.161	12.3
LRTC-TV	0.138	163.1	0.123	178.4	0.126	165.8	0.130	185.7	0.107	97.4	0.127	99.3	0.159	99.7	0.102	98.9
TMac	0.226	12.6	0.199	8.4	0.168	11.4	0.237	12.0	0.240	5.6	0.287	7.7	0.236	6.0	0.173	8.2
TMac-TT	0.225	21.0	0.191	28.6	0.181	28.1	0.203	20.7	0.165	10.2	0.215	9.6	0.201	7.4	0.145	9.8
TR-ALS	0.170	8.2	0.154	7.6	0.153	9.5	0.185	8.5	0.160	5.4	0.198	7.2	0.205	5.1	0.141	3.6
TTC	0.161	20.7	0.153	38.1	0.161	39.9	0.171	27.9	0.158	15.3	0.194	20.5	0.212	29.5	0.142	27.6
TTC-Tikhonov	0.161	21.3	0.153	39.5	0.160	40.1	0.170	28.1	0.158	15.3	0.194	20.6	0.212	30.1	0.142	27.4
TTC-TV	0.151	23.1	0.136	40.4	0.141	40.9	0.155	29.8	0.131	15.8	0.175	22.2	0.176	31.0	0.111	28.7

Fig. 5. Ground-truth of eight small benchmark images.

improved completion results compared to the standard TTC ALS method. Specifically, an initial λ_p of 1 is chosen throughout our later experiments. It is also possible to adjust the values of the λ_p parameters during the execution of Algorithm 1. As the estimate of the completed image improves over the iterations, it might not be necessary to keep enforcing the local smoothness. The heuristic we propose is to multiply the λ_p parameters with the relative error on the observed errors at the end of each iteration and use those values for the next iteration. In practice, this results in a more monotonic convergence of the relative error as a function of the iterations.

G. Tikhonov Regularization

In addition to the TV terms, Tikhonov regularization has also been considered in [18, p. 2213]. The tensor completion problem is then written as the following optimization problem

$$\min_{\mathcal{A}\in\mathcal{S}_{\mathrm{TT}}^{(d)}} ||\boldsymbol{S}^T \mathrm{vec}(\mathcal{A})-\boldsymbol{y}||_2^2+\sum_{p=1}^{P}\lambda_p||\mathcal{A}\times_p \boldsymbol{D}_p||_F^2+\gamma\,||\mathcal{A}||_F^2,$$
$$\text{s.t. } \mathcal{A}\in\mathcal{S}_{\mathrm{TT}}^{(d)}, \quad \text{and TT-rank}(\mathcal{A})=(R_1,R_2,\ldots,R_d). \tag{12}$$

Using the fact that the tensor train is in site-k-mixed-canonical form together with (3), the update step in the ALS algorithm is then modified to

$$(\boldsymbol{B}^T\boldsymbol{B}+\sum_{p=1}^{P}\lambda_p\boldsymbol{W}_p^T\boldsymbol{W}_p+\gamma\ \boldsymbol{I})\ \mathrm{vec}(\mathcal{A}^{(k)})=\boldsymbol{B}^T\boldsymbol{y}, \tag{13}$$

where $\boldsymbol{I}$ is an identity matrix of size $R_k I_k R_{k+1}$. The addition of a Tikhonov regularization term therefore comes at zero additional computational cost.

IV. EXPERIMENTAL RESULTS

In this section our proposed algorithm is compared extensively with state-of-the-art tensor completion methods in three experiments. The scalability of our algorithm in particular is demonstrated through the last two experiments. All experiments were run in MATLAB2018a on a desktop computer with an Intel i5 quad-core processor at 3.2GHz and 16GB RAM. A MATLAB implementation of Algorithm 1, together with necessary data files for reproducing all experimental results can be downloaded from https://github.com/IRENEKO/TTC. The values of all tuning parameters used in these experiments as well as benchmark images are all given in the supplemental materials. First, we apply Algorithm 1 to complete eight color images with approximate size 300×300 and compare its efficacy in runtime and completion accuracy with seven other state-of-the-art tensor completion algorithms. The two best methods from the first experiment are then compared with our algorithm in the second experiment for the completion of three color images with approximate size 4000×4000. The increased dimensions allow us to highlight the scalability of Algorithm 1 with these state-of-the-art methods. Another way to assess the scalability of our method is to apply it on higher order tensors. We therefore also compare Algorithm 1 with the two best methods from Experiment 1 to complete a color video. The completion accuracy of all methods is measured either by the relative standard error (RSE)

$$\mathrm{RSE}=\frac{||\mathcal{A}-\hat{\mathcal{A}}||_F}{||\mathcal{A}||_F},$$

or the peak signal-to-noise ratio (PSNR)

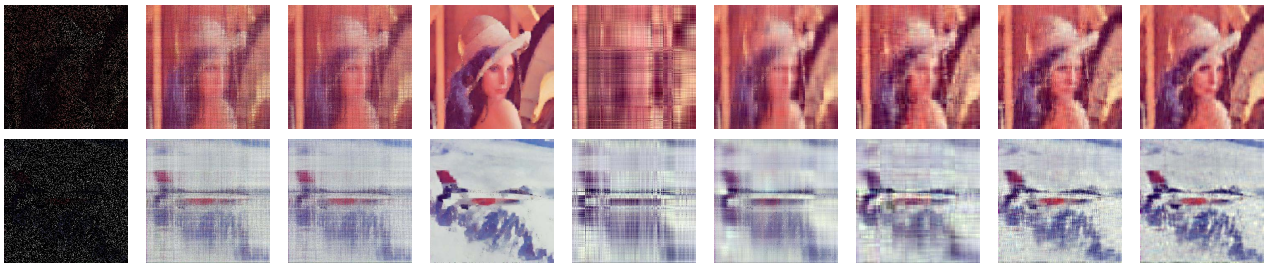
$$\mathrm{PSNR}=20\ \log_{10}(\mathrm{MAX_I})-10\ \log_{10}(\mathrm{MSE}),$$

where $\hat{\mathcal{A}}$ is the completed tensor, $\mathrm{MAX_I}$ is the maximum possible pixel value and MSE is the mean square error $||\mathcal{A}-\hat{\mathcal{A}}||_F/\mathrm{numel}(\mathcal{A})$, where $\mathrm{numel}(\mathcal{A})$ denotes the total number of entries in $\mathcal{A}$.

A. Small Image Inpainting

Eight benchmark images, shown in Figure 5, from the Berkeley Segmentation dataset[1] and USC-SIPI image database[2] were used to compare the performance of our proposed method with state-of-the-art completion algorithms in terms of completion accuracy and runtime. TTC denotes Algorithm 1 without any TV or Tikhonov regularization and TTC-TV denotes Algorithm 1 with TV regularization. Tikhonov regularization did not improve the results significantly in the task herein and its discussion is therefore conducted individually in Section IV-A.7. Table II lists the dimensions of the benchmark images and the dimension factorizations used for the tensor train methods. All images were resized using bicubic interpolation through the MATLAB "imresize" command. The eight

[1] https://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/
[2] http://sipi.usc.edu/database/database.php



(a) Observed (b) TNN (c) HaLRTC (d) LRTC-TV (e) TMac (f) TMac-TT (g) TR-ALS (h) TTC (i) TTC-TV

Fig. 6. Image inpaintings of *Lena*, and *Airplane* benchmarks by eight algorithms.

TABLE II

EXPERIMENT 1 DIMENSION SETTINGS

Image	Resized dimensions	Dimension factorization
House, River Bridge, Man	$324 \times 486 \times 3$	$9 \times 6 \times 6 \times 6 \times 9 \times 9 \times 3$
Lena, Peppers Baboon, Airplane	$300 \times 300 \times 3$	$10 \times 6 \times 5 \times 5 \times 6 \times 10 \times 3$

images are grouped into two groups with slightly different dimensions, which is a first attempt to determine how sensitive the runtime and completion accuracy of all methods are with respect to dimension size. Only 10% of each benchmark image was retained, whereby for each missing pixel all color information was removed. We fine-tune the hyper-parameters of the seven algorithms in comparison to give the best RSE scores on the images.

For the proposed TTC method, the TT-ranks are determined by cross-validation [38] on the completion error of a held-out 1% entries. That is, for an image with only 10% observed entries, $1/10$ of these known entries are kept for validation. We perform 10 trials in each cross-validation experiment and the TT-ranks that give the lowest average error of held-out entries are chosen. The same TT-ranks are shared by TTC-Tikhonov and TTC-TV experiments for the same image.

1) State-of-the-Art Methods: Algorithm 1 is compared with the following state-of-the-art methods: TNN,[3] HaLRTC, FaLRTC,[4] LRTC-TV,[5] TMac,[6] TMac-TT,[7] and TR-ALS.[8] These methods represent four different approaches towards the completion problem. The TNN method aims at minimizing the number of nonzeros in the tensor multi-rank, which is later relaxed to minimize the nuclear norm of a matrix constructed by frontal slices of the three way tensor [22]. HaLRTC, FaLRTC [4] and LRTC-TV [18] are on the other hand minimizing the sum of nuclear norms of the unfolded matrices. The TMac algorithm [7] includes two different schemes, TMac-inc and TMac-dec, depending on different rank adjustment strategies. Here we only compare with TMac-inc as it shows a better performance than TMac-dec in our experiments. The extension of TMac algorithm TMac-TT [28] is also considered. The TR-ALS algorithm [29] also uses a tensor train of $\mathcal{A}$ but with $R_1 > 1$. Moreover, it employs a different ALS for updating each tensor train core wherein each slice of the core is updated sequentially, unlike our way of updating the whole core at once. In these benchmarking algorithms, all MATLAB implementations written by the original authors were used.

[3]http://www.ece.tufts.edu/ shuchin/software.html

[4]http://www.cs.rochester.edu/u/jliu/code/TensorCompletion.zip

[5]https://xutaoli.weebly.com/

[6]http://www.math.ucla.edu/ wotaoyin/papers/codes/TMac.zip

[7]https://sites.google.com/site/jbengua/home/projects/efficient-tensor-completion-for-color-image-and-video-recovery-low-rank-tensor-train

[8]https://github.com/wangwenqi1990/TensorRingCompletion

TABLE III

AVERAGE RUNTIME OVER THE EIGHT IMAGES OF EACH ALGORITHM TO OBTAIN ITS BEST RSE SCORE AND THE CORRESPONDING AVERAGE RUNTIME OF THE TTC ALGORITHM TO OBTAIN THE SAME RSE SCORE

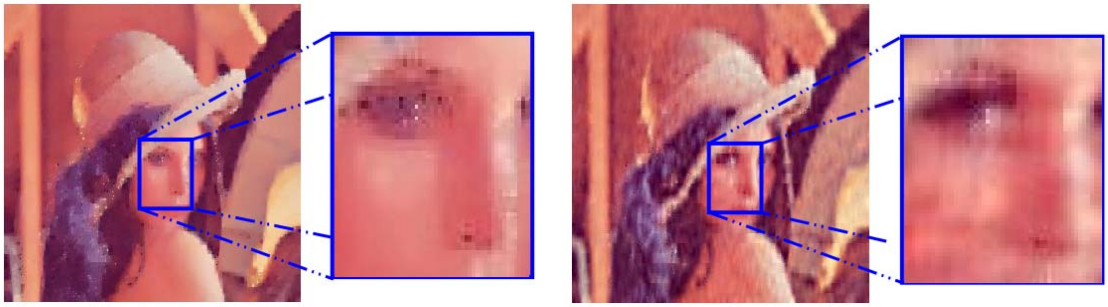
	TNN	HaLRTC	FaLRTC	TMac	TMac-TT	TR-ALS
Average runtime (s)	30.8	9.8	13.4	9.0	17.2	6.9
Average TTC runtime (s)	1.0	1.6	1.6	0.8	3.0	4.3
Speedup	30.8×	6.1×	8.4×	11.3×	5.7×	1.6×

2) Comparison With State-of-the-Art: The completion accuracy measured as the RSE of all algorithms and their corresponding runtimes are reported in Table I, while the completed images for *Lena* and *Airplane* are shown in Figure 6. The completed images obtained by the FaLRTC is indistinguishable from HaLRTC and has therefore been omitted from Figure 6. The proposed TTC-TV algorithm with TTC cross-validated TT-ranks outperforms the other six algorithms, excluding LRTC-TV, in terms of RSE for all eight benchmark images at the cost of an overall larger runtime than TR-ALS. Moreover, although TTC only reaches lower RSEs compared to its competitors on five out of eight benchmark images, we remark that it is able to obtain the same or even lower RSEs with less runtime with fine-tuned TT-ranks. Table III lists the runtime for each method averaged over all images to obtain its best RSE, together with the average runtime over all images for the TTC algorithm to obtain an identical RSE. The TTC algorithm is seen to be faster than the state-of-the-art methods, with average speedups of at least 10 compared with TMac and TNN. Moreover, it is worth noting that the proposed TTC-TV algorithm results in consistently smaller RSEs compared with those achieved by TTC at similar runtimes. These better RSE values obtained with TTC-TV result in completed images that are smoother while still preserving details. LRTC-TV, an extension of the LRTC methods with total variation terms, reaches the lowest RSEs in all benchmark images at the cost of taking 4.7 times longer than TTC-TV on average.

3) RSE Is Not Enough: Table I seems to indicate that LRTC-TV consistently obtains better completion results over all other methods. However, we argue that it is sometimes not enough to evaluate the completed images by their corresponding RSEs. A visual inspection of the completed images still

TABLE IV
PERFORMANCE (RSEs) OF FIVE ALGORITHMS ON EIGHT BENCHMARK IMAGES WITH 1% OBSERVED ENTRIES

Method	House		River		Bridge		Man		Lena		Peppers		Baboon		Airplane	
	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)
TNN	0.523	11.0	0.469	12.4	0.431	55.3	0.498	31.4	0.842	2.3	0.662	6.0	0.585	12.1	0.500	15.6
LRTC-TV	0.228	154.5	0.191	150.8	0.202	138.3	0.296	150.1	0.324	81.1	0.453	77.6	0.353	77.0	0.198	71.5
TMac-TT	0.360	19.4	0.278	18.7	0.241	17.8	0.326	17.1	0.301	9.6	0.409	20.9	0.292	11.5	0.209	9.3
TR-ALS	0.350	2.2	0.260	3.0	0.251	2.8	0.322	2.3	0.347	1.8	0.457	1.6	0.339	1.3	0.244	1.3
TTC-TV	0.210	3.2	0.197	5.7	0.199	1.4	0.232	2.4	0.232	0.9	0.295	0.7	0.250	1.1	0.176	1.0



(a) LRTC-TV: PSNR = 24.577dB, RSE = 0.107. (b) TTC-TV: PSNR = 24.423dB, RSE = 0.109.

Fig. 7. Image inpaintings of *Lena* by LRTC-TV and TTC-TV. LRTC-TV scores better on the PSNR and RSE metrics but lacks details due to oversmoothing.

remains the best way to compare results. We illustrate this point by comparing the completed *Lena* image using both the LRTC-TV and TTC-TV methods in Figure 7. Both the TT-rank and the number of iterations used in TTC-TV were increased to obtain an RSE score that is quite close to the LRTC-TV method. Although the LRTC-TV method has better RSE and PSNR scores, one can see that detailed features such as the eyes and feathers are blurred by the LRTC-TV method due to oversmoothing. This oversmoothing was observed for any nonzero value of the λ tuning parameters.

4) Influence of Smaller Portions of Observed Entries: In what follows, we explore the influence of smaller percentages of observed entries on the completion accuracy and runtime by recovering images with 1% observed. As before, we fine-tune the tuning parameters of different algorithms except the proposed TTC-TV, for which cross-validated hyper-parameters are used. LRTC-TV, TMac-TT and TTC-TV generally perform better than LRTCs, TMac and TTC, respectively, according to Table I, hence we omit the latter three in Table IV. LRTC-TV outperforms the TNN, TMac-TT and TR-ALS algorithms in five out of eight benchmark images at the cost of 6, 7, 55 times longer average runtime, respectively. TMac-TT reaches lower RSEs than LRTC-TV when completing the remaining three images. Moreover, the proposed TTC-TV outperforms all other methods in all benchmark images except *River*, where a similar RSE as that obtained by LRTC-TV is reached. It is also worthwhile to note that TTC-TV demonstrates a speedup of up to 55 times compared with LRTC-TV in the above investigation.

5) Effect of TT-Initialization on ALS Convergence: We also investigated whether our proposed initialization method results in better convergence behavior of ALS. An alternative initialization method, called the "missing entry zero filled" tensor initialization, is described in [29] and used in the TR-ALS method. Figure 8 depicts the RSEs of the completed *Man* benchmark image as a function of the iteration count using these two different initializations. If we assume that the RSE obtained after 8 iterations when using the proposed initialization method is taken as a threshold to stop the TTC algorithm, then the "missing entry zero-filled" tensor initialization would still not have converged after 20 iterations. In general, using our proposed initialization method always resulted in the RSE curve tapering off very fast over iterations. However, modifying the code for TNN, LRTC-TV, TMac, TMac-TT to the proposed initialization method is not trivial. For HaLRTC, FaLRTC and TR-ALS, the algorithms get stuck in local optima and achieve high RSEs ($\approx$ 0.9) with the interpolation initialization.

6) Influence of Different Factorization Sizes: In Section III-E, a general guideline for choosing a specific factorization size of a tensor is provided. Here we use the *House* benchmark image as an example to show the influence of different factorization sizes on both the obtained RSE, total runtime and problem size ($R_k I_k R_{k+1}$), all listed in Table V. The total number of iterations of Algorithm 1 is fixed for all factorizations. As shown in the table, similar RSEs are obtained when the maximal dimension in the factorization is limited to 9. A slight improvement of the runtime is observed when the number of tensor train cores is reduced from 13 down to 7 due to the trade-off between the number of cores and problem size as discussed in Section III-E. Both the RSE and total runtime are seen to degrade as the dimensions in the factorization are further increased, due to the fast growing problem size. These observations are consistent with the rules specified in Section III-E. Moreover, when we do not perform any dimension factorizations, we show in Table V that the condition number of the linear systems to-be-solved becomes infinitely large (due to the singularity of the linear system). Specifically, we point out that the condition number can also increase as we increase the TT-ranks adopted in TTC algorithms. We will illustrate this in the following Section IV-A.7.

7) Influence of Tihhonov regularization: Adding Tikhonov regularization term can be beneficial for solving linear systems

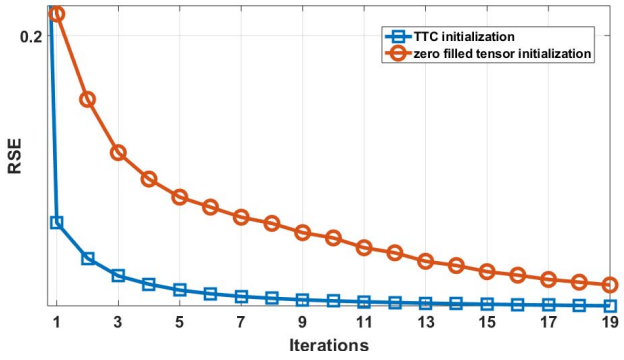


Fig. 8. The RSE as a function of the iteration count for two different initialization methods of the TTC algorithm.

TABLE V

INFLUENCE OF DIFFERENT FACTORIZATION SIZES

Dimension factorization	RSE	Time(s)	Problem size	Condition number
$324 \times 486 \times 3$	0.200	820.4	14580	∞
$18^2 \times 18 \times 27 \times 3$	0.174	28.4	2160	51.00
$6^2 \times 9 \times 6 \times 9^2 \times 3$	0.161	10.4	1683	71.45
$2^2 \times 3^4 \times 2 \times 3^5 \times 3$	0.161	14.4	1083	16.72

TABLE VI

SUPPLEMENTARY MRI SCANS DIMENSION SETTINGS

Image	Resized dimensions	Dimension factorization
Head-{1,2,3}		
Neck	512×512	$8 \times 8 \times 8 \times 8 \times 8 \times 8$

TABLE VII

PERFORMANCE (RSES) OF SEVEN ALGORITHMS ON FOUR BENCHMARK MRI IMAGES WITH 10% OBSERVED ENTRIES

Method	Head-1		Head-2		Head-3		Neck	
	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)	RSE	Time(s)
TNN	0.529	24.1	0.522	11.9	0.456	13.8	0.389	16.6
LRTC-TV	0.655	235.7	0.632	112.0	0.485	166.8	0.434	166.6
TMac-TT	0.541	7.6	0.531	3.7	0.489	3.0	0.443	1.3
TR-ALS	0.391	195.0	0.378	88.1	0.294	110.2	0.327	34.6
TTC	0.431	40.0	0.423	47.0	0.362	31.1	0.355	33.5
TTC-Tikhonov	0.428	42.0	0.421	47.3	0.360	31.6	0.353	33.4
TTC-TV	0.370	45.3	0.361	49.6	0.284	35.0	0.289	34.3

in two ways: it can help in regularizing the problem in Equation (13) for the case where the matrix $\boldsymbol{B}^{\boldsymbol{T}}\boldsymbol{B}$ in the ALS update is singular – adding a scaled unit matrix then makes the matrix in the normal equation invertible; it can also be interpreted as an equal scaling (down) of all pixels, which serves well in certain applications.

In Figure 9, we demonstrate that the linear systems become ill-conditioned (characterized by relatively large condition numbers) as the TT-ranks R_{mid} and R_{d-1} grow. When the condition number of original normal equation grows to 1118, the benefits of Tikhonov regularization become visible. On the other hand, we exploit the other gain by evaluating TTC, TTC-Tikhonov algorithms on completions of magnetic resonance imaging (MRI) scans of heads and necks[9] [39]. Table VI summarizes the dimensions of the benchmark images and the dimension factorizations considered. Since we show in the above experiments (Table I, IV, Figure 9) that TTC-TV is generally more powerful than TTC and TTC-Tikhonov, the TT-ranks herein are found by cross-validation using TTC-TV on a held-out 1% entries. The same TT-ranks are then shared by TTC and TTC-Tikhonov. By referring to Table VII, one

[9]https://www.cis.rit.edu/htbooks/mri/chap-14/chap-14.htm

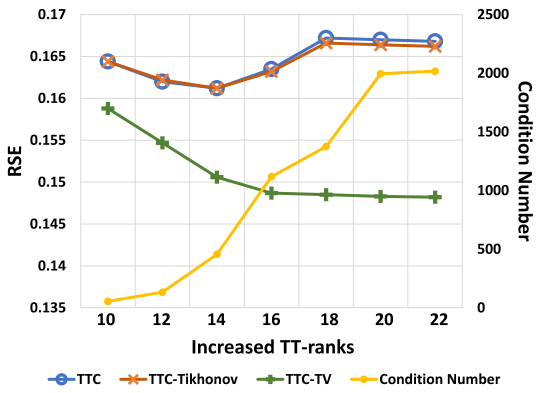


Fig. 9. The effect of increased TT-ranks to the condition numbers in linear systems and performance (RSEs) of TTC, TTC-Tikhonov, and TTC-TV algorithms.

TABLE VIII

EXPERIMENT 2 DIMENSION SETTINGS.

Image	Original dimensions	Dimension factorization
Dolphin	$4000 \times 3000 \times 3$	$2^5 \times 5^3 \times 2^3 \times 3 \times 5^3 \times 3$
Water Nature Fall	$6000 \times 4000 \times 3$	$2^4 \times 3 \times 5^3 \times 2^5 \times 5^3 \times 3$
Orion nebula	$6000 \times 6000 \times 3$	$2^4 \times 3 \times 5^3 \times 2^4 \times 3 \times 5^3 \times 3$

can see that, though not significant, the effect of Tikhonov regularization is discernible. We remark that the linear systems are well-conditioned with a condition number of 24.72 on average. Adding Tikhonov regularization, alternatively, helps to scale down the pixels by 1.43% with $\lambda_p = 0.001$ and reaches lower RSEs.

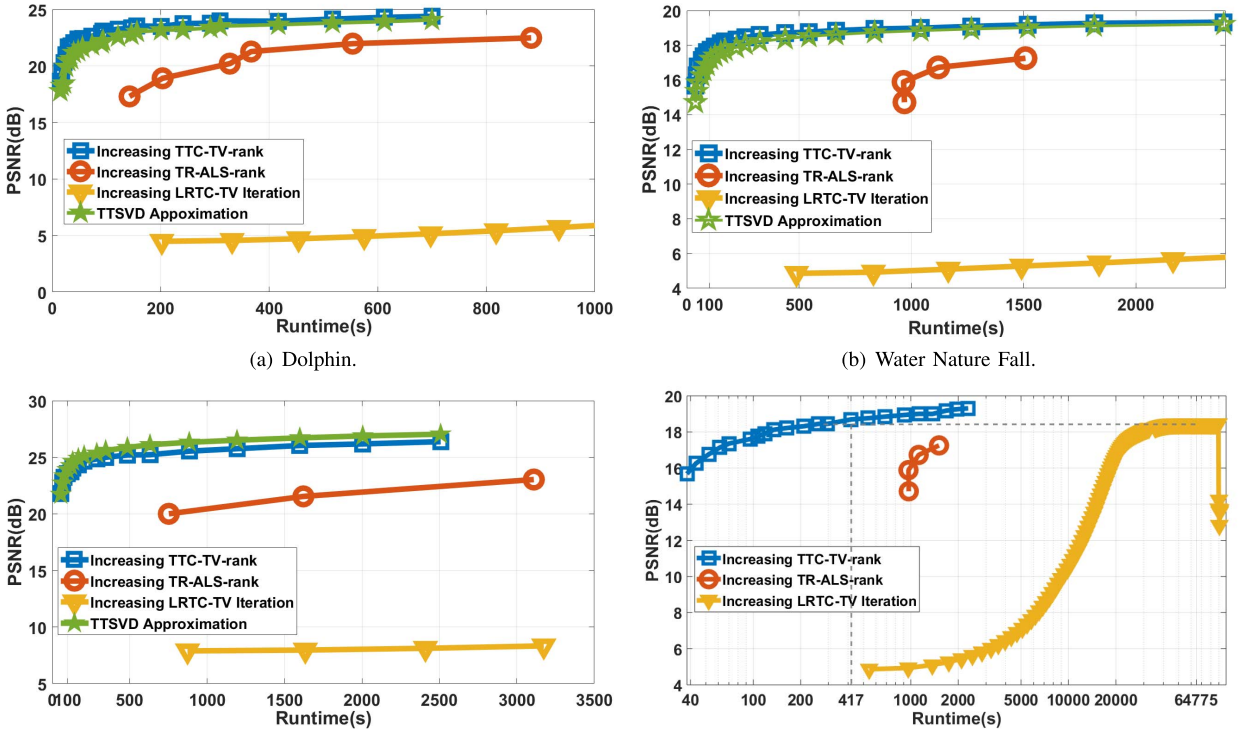
B. Large Image Inpainting

The experiments above have shown that the performance of both the TR-ALS and LRTC-TV methods are similar to the proposed TTC (TTC-TV) method. In this experiment we assess the scalability of the LRTC-TV, TR-ALS and TTC-TV methods by completing three high-resolution benchmark color images. The ground-truth of these three images is shown in the supplemental materials. The color images used in this section are *Dolphin*,[10] *Water Nature Fall*[11] and *Orion nebula*.[12] The dimensions of each image and their respective factorizations used in TTC-TV and TR-ALS are listed in Table VIII. The LRTC-TV method uses the original dimensions of each image. The corresponding tensor trains consisted of 16, 17 and 17 cores, respectively. Only 1% pixels of each image were retained. Figure 10 shows the PSNR obtained by TTC-TV, TR-ALS and LRTC-TV as a function of the total runtime for all three images. These graphs were constructed by increasing the TT-ranks for both the TTC-TV and TR-ALS methods, which resulted in better completion results at the cost of increased runtime per iteration. The TT-rank R_{mid} of the TTC-TV method was increased from 3 up to 24, 24 and 19 for *Dolphin*, *Water Nature Fall* and *Orion nebula*, respectively.

[10]http://absfreepic.com/free-photos/download/dolphin-4000 × 3000_21859.html

[11]http://absfreepic.com/freephotos/download/water-nature-fall-6000 × 4000_90673.html

[12]http://absfreepic.com/free-photos/download/orion-nebula-in-space-6000 × 6000_50847.html



(a) Dolphin. (b) Water Nature Fall.

(c) Orion nebula. (d) Extended runtime Water Nature Fall.

Fig. 10. Obtained PSNR of TTC-TV, TR-ALS and LRTC-TV on high-resolution benchmark images versus the total runtime.

(a) Original (b) LRTC-TV: PSNR=18 453dB (c) TTC-TV: PSNR=19 321dB

Fig. 11. Image inpaintings of *Water Natural Fall* by LRTC-TV and TTC-TV.

The TT-ranks could only be varied from 2 up to 7, 5 and 4 for the TR-ALS method, as the uniform TT-rank quickly lead to out-of-memory errors. This is reflected by the limited number of points in each of the TR-ALS graphs in Figure 10. The LRTC-TV method does not use tensor trains and we allowed it to run without time restriction.

All four figures in Figure 10 show that TR-ALS manages to achieve almost as good PSNR values as TTC-TV at about 10 times larger runtimes. The PSNR values obtained by the LRTC-TV method are unacceptable within reasonable runtimes. Figure 10(d) illustrates that LRTC-TV needs a runtime that is about **155 times** larger (64775 versus 417 seconds[13]) than TTC-TV to obtain the same PSNR on *Water Nature Fall*. This is due to the use of the Tucker decomposition and a corresponding computational complexity of $O(K(\prod_{k=1}^{d} I_k)^3)$, which scales badly with both d and I_k. We further note that LRTC-TV is unable to reach as high PSNR as TTC-TV in this experiment, and a sharp drop in the PSNR is witnessed after 89765 seconds as shown in Figure 10(d). The completed images of LRTC-TV and TTC-TV are depicted in Figure 11. By looking at the water flow in Figure 11(b), it is obvious that many missing pixels remain missing after LRTC-TV completion. As the RSE curves of these experiments lead to same conclusions as above, we only show the RSE curves of *Dolphin* benchmark in the supplemental materials.

Moreover, we further included the curve obtained by applying TT-SVD for given TT-ranks on the original benchmark images in Figure 11(a)(b)(c). Noted that for the given TT-ranks, the TT approximation given by TT-SVD method is quasi-optimal in the tensor-train subspace $\mathcal{S}_{\mathrm{TT}}^{(d)}$ [30, Corollary 2.4.]. Thus including the curves enables us to evaluate the gaps between the result obtained by TTC-TV and a quasi-optimal solution in the subspace. However, we stress that the curve is only meaningful when comparing to the curve of TTC-TV, because the subspaces of TR-ALS and LRTC-TV are

[13]6011 versus 39 seconds on *Dolphin* to reach PSNR = 22dB and 59985 versus 345 seconds on *Orion nebula* to reach PSNR = 25dB.

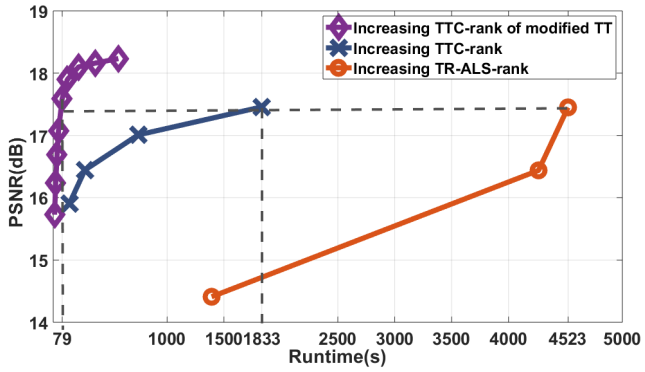


Fig. 12. Obtained PSNR of both TTC and TR-ALS on the video versus the total runtime for increasing TT-ranks.

essentially different from that of TTC-TV. We increased the TT-rank R_{mid} of the TT-SVD approximation exactly the same as we did for TTC-TV, and compare the PSNRs of TTC-TV and TT-SVD approximation for same the TT-ranks. It is then seen that the result obtained by TTC-TV is nearly or even better than a quasi-optimal approximation in the tensor train subspace $\mathcal{S}_{\text{TT}}^{(d)}$.

C. Color Video Completion

The inpainting of high-resolution images demonstrated the lack of scalability of the LRTC-TV method. This method will therefore not be considered anymore. The purpose of this experiment is to demonstrate an advantageous feature of our proposed TTC algorithm with regard to color video completion. A 144 frames video clip was taken from the *Mariano Rivera Ultimate Career Highlights* video, available on YouTube.[14] The dimensions of the resulting tensor and their factorizations are given in Table IX, resulting in a tensor train of eleven cores. The total number of elements of this particular tensor has the same order of magnitude as for the high-resolution images. In this experiment 10% of the video is retained. Observed pixels appear consistently at the same position over all frames and color channels, which models a breakdown of 90% of the available sensors in the camera. The fact that all observed pixels occur at the same position over all frames can be taken into account by solving the following optimization problem

$$\min_{\boldsymbol{A}} \; ||\boldsymbol{S}^T \boldsymbol{A} - \boldsymbol{Y}||_F^2, \;\; \text{s.t. TT-rank}(\boldsymbol{A}) = (R_1, \ldots, R_d), \quad (14)$$

where $\boldsymbol{A}$ is the original tensor reshaped into a $(360 \cdot 640) \times (144 \cdot 3)$ matrix. This implies that the observed values are also reshaped into the matrix $\boldsymbol{Y}$. The input matrices related to the frame and color dimensions are in this way not necessary anymore. The matrix $\boldsymbol{A}$ is then modeled by a tensor train of seven cores for which $\boldsymbol{\mathcal{A}}^{(1)}$ has dimensions $1 \times 9 \times (144 \cdot 3) \times R_2$. The $(144 \cdot 3)$ dimension of $\boldsymbol{\mathcal{A}}^{(1)}$ corresponds with the columns of the $\boldsymbol{A}$ matrix and updating this TT-core is done by rewriting (8) into a matrix equation. All other TT-cores are updated by solving (8). Figure 12 shows the PSNR as a function of the total runtime obtained by the conventional TTC algorithm, its modification (14), and TR-ALS. The TT-rank for the TR-ALS method was varied from 2 up to 4, higher values for the ranks resulted in out-of-memory-errors. The best PSNR obtained with TR-ALS is 17.45 dB and is obtained after a total runtime of 4523 seconds. The conventional TTC algorithm was run for a fixed number of 3 iterations and increasing TT-ranks from 2 up to 5 and obtains the same PSNR value about 2.5 times faster. The modified TTC algorithm for solving Equation (14) was run for a fixed number of 3 iterations and increasing TT-ranks from 2 up to 10. The smaller amount of TT-cores results in faster runtimes and better PSNRs. The best PSNR value obtained by TR-ALS is obtained by this particular TTC implementation in 79 seconds, about 57 times faster than TR-ALS and 23 times faster than the standard TTC formulation.

[14]https://www.youtube.com/watch?v=UPtDJuJMyhc

TABLE IX
EXPERIMENT 3 DIMENSION SETTINGS

Original dimensions	Dimension factorization
$360 \times 640 \times 144 \times 3$	$9 \times 8 \times 5 \times 4 \times 4 \times 5 \times 8 \times 4 \times 6 \times 6 \times 3$

V. CONCLUSION

We have proposed an efficient tensor completion framework by assuming tensor train structures in the underlying regression model. Specifically, the multi-indices (coordinates) of the known entries act as inputs and their corresponding values act as outputs. Moreover, Total Variation regularization and Tikhonov regularization are readily realized under the tensor train framework with almost zero additional computational cost. A simple yet effective tensor train initialization method based on interpolations has also been introduced for images and videos. Extensive experiments with low percentages of known pixels have shown that the proposed algorithm not only outperforms the state-of-the-art methods in both accuracy and time cost, but also demonstrates better scalability.

REFERENCES

[1] Z. Xing, M. Zhou, A. Castrodad, G. Sapiro, and L. Carin, "Dictionary learning for noisy and incomplete hyperspectral images," *SIAM J. Imag. Sci.*, vol. 5, no. 1, pp. 33–56, Jan. 2012.
[2] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, "Tensor robust principal component analysis: Exact recovery of corrupted low-rank tensors via convex optimization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5249–5257.
[3] K. A. Patwardhan, G. Sapiro, and M. Bertalmio, "Video inpainting under constrained camera motion," *IEEE Trans. Image Process.*, vol. 16, no. 2, pp. 545–553, Feb. 2007.
[4] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, Jan. 2013.
[5] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, vol. 27, no. 2, Feb. 2011, Art. no. 025010.
[6] H. Fang, Z. Zhen, Y. Shao, and C.-J. Hsieh, "Improved bounded matrix completion for large-scale recommender systems," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI)* Melbourne, VIC, Australia: AAAI Press, Aug. 2017, pp. 1654–1660.
[7] Y. Xu *et al.*, "Parallel matrix factorization for low-rank tensor completion," *Inverse Problems Imag.*, vol. 9, no. 2, pp. 601–624, 2015.
[8] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Aug. 2009.
[9] L. Grasedyck, M. Kluge, and S. Krämer, "Variants of alternating least squares tensor completion in the tensor train format," *SIAM J. Sci. Comput.*, vol. 37, no. 5, pp. A2424–A2450, Jan. 2015.
[10] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering," in *Proc. 4th ACM Conf. Recommender Syst. (RecSys)*, 2010, pp. 79–86.

[11] B. Romera-Paredes, H. Aung, N. Bianchi-Berthouze, and M. Pontil, "Multilinear multitask learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1444–1452.
[12] M. Collins and S. B. Cohen, "Tensor decomposition for fast parsing with latent-variable PCFGs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2519–2527.
[13] P. Liu, X. Qiu, and X. Huang, "Learning context-sensitive word embeddings with neural tensor skip-gram model," in *Proc. 24th Int. Conf. Artif. Intell. (IJCAI)*, 2015, pp. 1284–1290.
[14] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Found. Comput. Math.*, vol. 9, no. 6, p. 717, Dec. 2009.
[15] Y. Chen, "Incoherence-optimal matrix completion," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2909–2923, May 2015.
[16] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm," *Math. Program. Comput.*, vol. 4, pp. 333–361, 2012.
[17] M. Signoretto, Q. Tran Dinh, L. De Lathauwer, and J. A. K. Suykens, "Learning with tensors: A framework based on convex optimization and spectral regularization," *Mach. Learn.*, vol. 94, no. 3, pp. 303–351, Mar. 2014.
[18] X. Li, Y. Ye, and X. Xu, "Low-rank tensor completion with total variation for visual data inpainting," in *Proc. AAAI*, 2017, pp. 2210–2216.
[19] H. Tan, B. Cheng, W. Wang, Y.-J. Zhang, and B. Ran, "Tensor completion via a multi-linear low-n-rank factorization model," *Neurocomputing*, vol. 133, pp. 161–169, Jun. 2014.
[20] T.-Y. Ji, T.-Z. Huang, X.-L. Zhao, T.-H. Ma, and G. Liu, "Tensor completion using total variation and low-rank matrix factorization," *Inf. Sci.*, vol. 326, pp. 243–257, Jan. 2016.
[21] M. E. Kilmer, K. Braman, N. Hao, and R. C. Hoover, "Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 1, pp. 148–172, Jan. 2013.
[22] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer, "Novel methods for multilinear data completion and de-noising based on tensor-SVD," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 3842–3849.
[23] C. Mu, B. Huang, J. Wright, and D. Goldfarb, "Square deal: Lower bounds and improved relaxations for tensor recovery," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 73–81.
[24] T. Suzuki, "Convergence rate of Bayesian tensor estimator and its minimax optimality," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1273–1282.
[25] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian CP factorization of incomplete tensors with automatic rank determination," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1751–1763, Sep. 2015.
[26] M. Imaizumi, T. Maehara, and K. Hayashi, "On tensor train rank minimization: Statistical efficiency and scalable algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3933–3942.
[27] H. N. Phien, H. D. Tuan, J. A. Bengua, and M. N. Do, "Efficient tensor completion: Low-rank tensor train," 2016, *arXiv:1601.01083*. [Online]. Available: http://arxiv.org/abs/1601.01083
[28] J. A. Bengua, H. N. Phien, H. D. Tuan, and M. N. Do, "Efficient tensor completion for color image and video recovery: Low-rank tensor train," *IEEE Trans. Image Process.*, vol. 26, no. 5, pp. 2466–2479, May 2017.
[29] W. Wang, V. Aggarwal, and S. Aeron, "Efficient low rank tensor ring completion," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5697–5705.
[30] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Jan. 2011.
[31] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM J. Scientific Comput.*, vol. 34, no. 2, pp. A683–A713, Jan. 2012.
[32] T. Rohwedder and A. Uschmajew, "On local convergence of alternating schemes for optimization of convex problems in the tensor train format," *SIAM J. Numer. Anal.*, vol. 51, no. 2, pp. 1134–1162, Jan. 2013.
[33] Q. Zhao, M. Sugiyama, and A. Cichocki, "Learning efficient tensor representations with ring structure networks," *CoRR*, vol. abs/1705.08286, pp. 1–13, May 2017. [Online]. Available: http://arxiv.org/abs/1705.08286
[34] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Ann. Phys.*, vol. 349, pp. 117–158, Oct. 2014.
[35] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Ann. Phys.*, vol. 326, no. 1, pp. 96–192, 2011, Jan. 2011.
[36] K. Batselier, Z. Chen, and N. Wong, "Tensor network alternating linear scheme for MIMO volterra system identification," *Automatica*, vol. 84, pp. 26–35, Oct. 2017.
[37] I. V. Oseledets, "Approximation of $2^d \times 2^d$ matrices using tensor decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 2130–2145, Jan. 2010.
[38] C.-G. Li and R. Vidal, "A structured sparse plus structured low-rank framework for subspace clustering and completion," *IEEE Trans. Signal Process.*, vol. 64, no. 24, pp. 6557–6570, Dec. 2016.
[39] J. Zhang *et al.*, "HF-SENSE: An improved partially parallel imaging using a high-pass filter," *BMC Med. Imag.*, vol. 19, no. 1, p. 27, Dec. 2019.