IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 6, 2020, accepted February 25, 2020, date of publication February 28, 2020, date of current version March 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977050

Tensor Completion Methods for Collaborative Intelligence

LIOR BRAGILEVSKY, (Student Member, IEEE), AND IVAN V. BAJIĆ, (Senior Member, IEEE)
School of Engineering Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada

Corresponding author: Lior Bragilevsky (lbragile@sfu.ca)

This work was supported in part by the Alexander Graham Bell Canada Graduate Scholarship-Master's (CGS-M), and in part by the Natural Sciences and Engineering Research Council (NSERC) under Grant RGPIN-2016-04590.

ABSTRACT In the race to bring Artificial Intelligence (AI) to the edge, *collaborative intelligence* has emerged as a promising way to lighten the computation load on edge devices that run applications based on Deep Neural Networks (DNNs). Typically, a deep model is split at a given layer into edge and cloud sub-models. The deep feature tensor produced by the edge sub-model is transmitted to the cloud, where the remaining computationally intensive workload is performed by the cloud sub-model. The communication channel between the edge and cloud is imperfect, which will result in missing data in the deep feature tensor received at the cloud side, an issue that has mostly been ignored by existing literature on the topic. In this paper we study four methods for recovering missing data in the deep feature tensor. Three of the studied methods are existing, generic tensor completion methods, and are adapted here to recover deep feature tensor data, while the fourth method is newly developed specifically for deep feature tensor completion. Simulation studies show that the new method is $3-18$ times faster than the other three methods, which is an important consideration in collaborative intelligence. For VGG16's sparse tensors, all methods produce statistically equivalent classification results across all loss levels tested. For ResNet34's non-sparse tensors, the new method offers statistically better classification accuracy (by $0.25\%-6.30\%$) compared to other methods for matched execution speeds, and second-best accuracy among the four methods when they are allowed to run until convergence.

INDEX TERMS Tensor completion, missing data imputation, tensor decomposition, tensor reconstruction, collaborative intelligence, deep feature transmission, deep learning.

I. INTRODUCTION

As the Internet of Things (IoT) infrastructure gets deployed, there will be many opportunities for innovative applications that make use of the newly available sensor data. Many of these new applications will rely on DNNs to process the sensor data and produce useful predictions and analytics. One current research direction is towards miniaturization of DNNs, so that they can be implemented at or near the edge sensors, with limited computation and energy resources. While such DNNs may be sufficient for certain applications, resources available in the cloud will always be able to support larger and more sophisticated models than those that could be deployed at the edge. Hence, cloud-based analytics will remain essential even if some of the AI-based processing gets moved to the edge.

The associate editor coordinating the review of this manuscript and approving it for publication was Juan Wang.

Collaborative Intelligence (CI) [1]–[4] is an AI deployment strategy that leverages both edge-based and cloud-based resources to make DNN computing faster and more efficient. In CI, a deep model is split into an edge sub-model and a cloud sub-model. For example, an edge sub-model may consist of the initial m layers of a DNN, while the cloud sub-model is made up of the remaining DNN layers. When an input signal is captured by an edge sensor, the edge sub-model processes the signal and produces a tensor of deep features, which is then transmitted to the cloud for subsequent processing by the cloud sub-model. Due to the imperfect channel between the edge and the cloud, deep feature tensor data may be damaged or missing, much like data transmitted over the Internet. Hence, error control schemes must be deployed to achieve seamless operation of edge and cloud sub-models in AI.

In this paper, we study four methods for recovery of missing data in a deep feature tensor. Three of these methods

come from existing literature: Simple Low Rank Tensor Completion (SiLRTC) [5], High Accuracy Low Rank Tensor Completion (HaLRTC) [5], and the recent Fused Canonical Polyadic (FCP) decomposition [6]. All three are general tensor completion methods based on the low-rank tensor assumption. We explain how these methods can be adapted to deep feature tensor completion. The fourth method we study is simple and tailor-made for recovery of missing data (i.e., imputation) in deep feature tensors. It is *adaptive* and *linear* in nature. Specifically, missing data in the tensor are recovered as a linear combination of other, available data, so the tensor rank is not increased. Due to these attributes, we call it Adaptive Linear Tensor Completion (ALTeC).

The paper is organized as follows. Section II provides the background that is necessary to understand the tensor completion algorithms under study. In Section III, we review the SiLRTC, HaLRTC, and FCP algorithms and explain how they are applied to deep feature tensor completion. Section IV introduces ALTeC and describes how its parameters are computed. Section V describes the simulation environment and experiments, and provides statistical analysis of the results. Lastly, Section VI concludes the paper and indicates potential avenues for further work.

II. PRELIMINARIES

In this section we illustrate several tensor and matrix operations that are used later in the paper. In terms of notation, bold calligraphic letters ($\boldsymbol{\mathcal{X}}$) will denote tensors, bold uppercase non-italic letters ($\mathbf{X}$) will denote matrices, bold lowercase non-italic letters ($\mathbf{x}$) will denote vectors, and italic letters (x or X) will denote scalars.

A. TENSOR FOLDING & UNFOLDING

Tensor *unfolding* is a structured mapping from a tensor to a matrix. A tensor can be unfolded along any of its dimensions. For example, consider a 3D tensor $\boldsymbol{\mathcal{X}}$ with two channels,

$$\boldsymbol{\mathcal{X}} = \left(\begin{bmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{bmatrix} \right), \tag{1}$$

where the left matrix (even integers) is the first channel and the right matrix (odd integers) is the second channel. Then unfolding along the x-, y-, and z-axis (axis 0, 1, and 2) produces matrices $\mathbf{X}_0$, $\mathbf{X}_1$, and $\mathbf{X}_2$, respectively, shown below.

$$\begin{aligned} \mathbf{X}_0 &= \text{unfold}(\boldsymbol{\mathcal{X}}, 0) = \begin{bmatrix} 0 & 2 & 4 & 1 & 3 & 5 \\ 6 & 8 & 10 & 7 & 9 & 11 \\ 12 & 14 & 16 & 13 & 15 & 17 \end{bmatrix} \\ \mathbf{X}_1 &= \text{unfold}(\boldsymbol{\mathcal{X}}, 1) = \begin{bmatrix} 0 & 6 & 12 & 1 & 7 & 13 \\ 2 & 8 & 14 & 3 & 9 & 15 \\ 4 & 10 & 16 & 5 & 11 & 17 \end{bmatrix} \\ \mathbf{X}_2 &= \text{unfold}(\boldsymbol{\mathcal{X}}, 2) \\ &= \begin{bmatrix} 0 & 6 & 12 & 8 & 14 & 4 & 10 & 16 \\ 1 & 7 & 13 & 9 & 15 & 5 & 11 & 17 \end{bmatrix} \end{aligned} \tag{2}$$

Once a 3D tensor is unfolded into a 2D matrix, then matrix operations such as Singular Value Decomposition (SVD) can be performed. Once matrix processing is done, *folding* operation converts the 2D matrix into a 3D tensor. Folding along a given axis is the inverse of unfolding along the same axis, i.e., $\text{fold}(\text{unfold}(\boldsymbol{\mathcal{X}}, i), i) = \boldsymbol{\mathcal{X}}$.

B. SINGULAR VALUE DECOMPOSITION & SHRINKAGE

Singular Value Decomposition (SVD) of a given $m \times n$ matrix $\mathbf{A}$ is given by [7]:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices ($\mathbf{U}\mathbf{U}^T = \mathbf{I}, \mathbf{V}\mathbf{V}^T = \mathbf{I}$) whose dimensions are $m \times m$ and $n \times n$, respectively. The matrix Σ is $m \times n$ and contains the singular values of $\mathbf{A}$ along its main diagonal, $\Sigma = diag(\sigma_i)$.

In some applications, soft-thresholding (also known as *shrinkage*) [8] is applied to singular values in Σ in order to arrive at a lower-rank matrix that is still a good approximation to the original matrix $\mathbf{A}$. Specifically, if σ_i are the singular values of $\mathbf{A}$ and τ is a given threshold, then the corresponding shrinkage operation is defined as

$$\text{shrink}(\mathbf{A}, \tau) = \mathbf{U}\Sigma_\tau\mathbf{V}^T, \tag{4}$$

where $\Sigma_\tau = diag(\max(\sigma_i - \tau, 0))$. In words, shrinkage reduces all singular values by τ, clips the negative results to 0, and then re-synthesizes the matrix with the new singular values.

C. CANONICAL POLYADIC DECOMPOSITION

A $m \times n \times p$ tensor $\boldsymbol{\mathcal{X}}$ can be approximated with a low-rank tensor $\widehat{\boldsymbol{\mathcal{X}}}$ through Canonical Polyadic Decomposition (CPD) by solving [9]:

$$\min_{\widehat{\boldsymbol{\mathcal{X}}}} \left\| \boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}} \right\|_F, \tag{5}$$

where $\|\cdot\|_F$ is the Frobenius norm of a tensor, and the low-rank approximation tensor, $\widehat{\boldsymbol{\mathcal{X}}}$, is given by:

$$\widehat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{\text{rank}(\boldsymbol{\mathcal{X}})} \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r = [\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!] \tag{6}$$

The dimensions of $\mathbf{u}_r$, $\mathbf{v}_r$, and $\mathbf{w}_r$ are $m \times 1$, $n \times 1$, and $p \times 1$, respectively. The symbol $\circ$ represents the vector outer product, such that $\mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$ is a $m \times n \times p$ tensor for any r. The matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ represent the CP factor matrices whose columns are the corresponding vectors $\mathbf{u}_r$, $\mathbf{v}_r$, and $\mathbf{w}_r$, namely, $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{\text{rank}(\boldsymbol{\mathcal{X}})}]$, $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_{\text{rank}(\boldsymbol{\mathcal{X}})}]$, and $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_{\text{rank}(\boldsymbol{\mathcal{X}})}]$. The above minimization problem can also be rephrased and solved in terms of the unfolded tensors $\widehat{\boldsymbol{\mathcal{X}}}_{(i)}$ for $i = 1, 2, 3$:

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{X}}}_{(1)} &= (\mathbf{W} \odot \mathbf{V})\mathbf{U}^T \\ \widehat{\boldsymbol{\mathcal{X}}}_{(2)} &= (\mathbf{W} \odot \mathbf{U})\mathbf{V}^T \\ \widehat{\boldsymbol{\mathcal{X}}}_{(3)} &= (\mathbf{V} \odot \mathbf{U})\mathbf{W}^T, \end{aligned} \tag{7}$$

where the symbol $\odot$ represents the Khatri-Rao product defined as $\mathbf{U} \odot \mathbf{V} = [\mathbf{u}_1 \otimes \mathbf{v}_1, \ldots, \mathbf{u}_{\text{rank}(\boldsymbol{\mathcal{X}})} \otimes \mathbf{v}_{\text{rank}(\boldsymbol{\mathcal{X}})}]$, and $\otimes$ is the Kronecker product of the corresponding column-vectors.

D. TENSOR DATA PACKETIZATION & TRANSMISSION

In collaborative intelligence (CI), tensor data needs to be transferred from the edge to the cloud. This process involves writing tensor values into data packets (which we refer to as *packetization*) and sending these data packets over the network to the cloud. It is too early to say what kind of tensor packetization schemes will be adopted in CI applications in the future. We note, however, that transmission of another kind of tensor data, namely video, has been around for a while, and video packetization schemes are well-established and tested in practice. We believe that similar schemes will be strong contenders for deep tensor data packetization as well. Therefore, for the purposes of this paper, we adopt a packetization method that is popular in video streaming [10], where in each video frame, rows of macro-blocks are mapped to packets. In a similar manner, we write tensor data into packets row-by-row and then channel-by-channel. For example, the data in tensor $\mathcal{X}$ in (1) would generate six packets, each composed of one row of tensor data: [0, 2, 4], [6, 8, 10], . . . , [13, 15, 17].

Errors in the communication channel may cause data packets not to arrive at the cloud sub-model. Such packets are referred to as ''lost'' and the probability of their loss is p_{loss}.[1] The result of packet loss at the cloud side is that the corresponding tensor rows are not available. As an example, tensor $\mathcal{X}$ from (1), with two rows of missing data, is shown below:

$$\widetilde{\mathcal{X}} = \left(\begin{bmatrix} 0 & 2 & 4 \\ ? & ? & ? \\ 12 & 14 & 16 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ ? & ? & ? \end{bmatrix} \right), \tag{8}$$

where the missing values are indicated by question marks ('?'). The goal of tensor completion is to recover these missing values so that the cloud sub-model can perform successful inference.

III. GENERAL TENSOR COMPLETION ALGORITHMS

Tensor completion has found applications in a number of research areas, including computer vision, data analytics, etc. To our knowledge, however, it has not been studied in the context of recovering missing feature tensor values in collaborative intelligence. Often, the underlying assumption is that tensor data is ''low-rank'', or, more generally, ''smooth'' in some sense. A number of methods [5], [11], [12] have been developed based on the assumption that tensors lie in a low-rank manifold, which leads to iterative procedures for approximating the original tensor by a low-rank tensor. In these cases, it is not crucial to know where the data comes from, so long as the low-rank assumption holds. We refer to these methods as *general*, meaning that they could be applied to any kind of tensor. We will review three such methods, namely Simple Low Rank Tensor Completion (SiLRTC) [5], High Accuracy Low Rank Tensor Completion (HaLRTC) [5], and Fused Canonical Polyadic (FCP) [6], and adapt them to the problem of recovering missing feature tensor values produced by packet loss in collaborative intelligence.

We note that it is not clear whether in fact low-rank assumption holds for deep feature tensors. The existence of adversarial examples [13] shows that small perturbations in the input of a deep model may cause large changes downstream, which might indicate that the notions of smoothness and low-rank are less applicable to the deep feature tensors than they might be to the kind of data for which tensor completion has mostly been used so far, such as color images. Nonetheless, it is still important to establish what level of performance existing tensor completion methods can achieve on this new problem.

A. SIMPLE LOW RANK TENSOR COMPLETION (SILRTC)

In this section, we briefly review SiLRTC [5] and show how it can be applied to completion of feature tensors in collaborative intelligence. A summary of SiLRTC is shown in Algorithm 1 in the Appendix. The inputs are the corrupt tensor $\widetilde{\mathcal{X}}$ (with some of its rows missing), the number of iterations K, and non-negative scaling factors $\alpha_i, \beta_i, i \in \{1, 2, 3\}$ where α_i's add up to 1. The scaling factors are chosen randomly [5], and each pair of (α_i, β_i) is for one dimension of the tensor. The output is the ''completed'' tensor $\widehat{\mathcal{X}}$.

To reconstruct the missing values in $\widetilde{\mathcal{X}}$, SiLRTC makes a copy of $\widetilde{\mathcal{X}}$ in $\widehat{\mathcal{X}}$ (step 1) and then loops over K iterations. In each iteration, $\widehat{\mathcal{X}}$ is unfolded along each dimension (step 5), shrinking with threshold $\tau = \alpha_i/\beta_i$ is applied (step 6), and the tensor is folded back to 3D (step 7). The three folded tensors are added (step 8) and the result is re-scaled (step 10). The above operations change all values in the tensor, however, some of the rows in $\widetilde{\mathcal{X}}$ are known (i.e., not missing). Therefore, known rows are replaced in the resulting tensor (step 11) and the result is passed to the next iteration.

In essence, SiLRTC attempts to iteratively reduce the rank of the corrupt tensor $\widetilde{\mathcal{X}}$ by performing shrinkage on the unfolded versions of the tensor and averaging the folded results. The next method we review, HaLRTC, is based on a similar idea, but is more accurate.

B. HIGH ACCURACY LOW RANK TENSOR COMPLETION (HALRTC)

HaLRTC [5] follows the same reasoning as SiLRTC, but using an alternating direction method of multipliers (ADMM) [14] to find the solution. It is more sophisticated than SiLRTC and is expected to produce better tensor completion results. The steps are presented in Algorithm 2 in the Appendix.

The inputs are the corrupt tensor $\widetilde{\mathcal{X}}$ (with some of its rows missing), the number of iterations K, and non-negative scaling factors $\rho, \alpha_i, i \in \{1, 2, 3\}$ where α_i's add up to 1. The scaling factors are chosen randomly [5], and each α_i is for one dimension of the tensor. The output is the ''completed'' tensor $\widehat{\mathcal{X}}$.

[1] This probability depends on many factors, including physical layer modulation and error control, noise and interference in the channel, network congestion, etc.

The algorithm starts by initializing the output tensor $\widehat{\mathcal{X}}$ (step 1) and three auxiliary tensors $\mathcal{M}_i, \mathcal{Y}_i$ (step 3), one for each dimension. Then, in each of the K iterations, for each tensor dimension, $\widehat{\mathcal{X}}$ and $\mathcal{Y}_i$ are unfolded along the corresponding dimension (steps 7-8), their sum is shrunk (step 9) and the result folded into $\mathcal{M}_i$ (step 10). Finally, output tensor $\widehat{\mathcal{X}}$, auxiliary tensors $\mathcal{Y}_i$, and scaling parameter ρ are updated (steps 12, 15, and 17, respectively).

C. FUSED CANONICAL POLYADIC (FCP) DECOMPOSITION

The FCP algorithm [6] is somewhat more involved than SiLRTC and HaLRTC. The main steps are presented in Algorithm 3 in the Appendix, but the reader is referred to [6] for full details. The inputs to the algorithm include the corrupt tensor $\widetilde{\mathcal{X}}$ (with some of its rows missing), the CP factor matrices $\mathbf{A}_i, i \in \{1, 2, 3\}$ (which represent $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ in (6)), regularization matrices $\mathbf{L}_s$ and $\mathbf{L}_p$, the number of iterations K, non-negative scaling factors $\mu, \alpha_i, \beta_i, \delta_i, \zeta_i, i \in \{1, 2, 3\}$, tensor rank R along with its corresponding increment (R_u) and maximum value (R_m), and stopping criteria variables η and ϵ. The output is the "completed" tensor $\widehat{\mathcal{X}}$.

The algorithm starts by initializing the output tensor $\widehat{\mathcal{X}}$ (step 1). Then it loops (*iter*) through each of the K iterations and for each tensor dimension, $\widehat{\mathcal{X}}$ is unfolded along the corresponding dimension i and matrix $\mathbf{S}$ is computed (step 4) as the Khatri-Rao product between a pair of CP factor matrices (depending on i) as follows:

$$\mathbf{S} = \begin{cases} \mathbf{A}_3 \odot \mathbf{A}_2 & \text{for } i = 1 \\ \mathbf{A}_1 \odot \mathbf{A}_2 & \text{for } i = 2 \\ \mathbf{A}_2 \odot \mathbf{A}_1 & \text{for } i = 3 \end{cases} \tag{9}$$

Then the reciprocal of the step size L and the gradient of regularized error ∇h are computed (steps 5-6). The gradient computation involves the rectified linear function $Q(\cdot)$ defined as

$$Q(x) = \begin{cases} -1 & \text{for } x < -1 \\ x & \text{for } -1 \le x \le 1 \\ 1 & \text{for } x > 1 \end{cases} \tag{10}$$

After that, the CP factor matrices $\mathbf{A}_i$ are updated in steps 9-12 until the stopping criteria (computed by the function stopping_criterion$(\cdot)$ in step 8) are met. It should be noted that stopping criteria, as well as several other parameters and scaling factors, are different for sparse and non-sparse tensors [6]. Next, the tensor rank is updated if needed (steps 15-17), an operation is performed on the CP factor matrices and the result is folded back into $\widehat{\mathcal{X}}^{iter-1}$ along the first tensor dimension (step 18). Lastly, $\widehat{\mathcal{X}}^{iter-1}$ is assigned to $\widehat{\mathcal{X}}^{iter}$ (step 19), $\widehat{\mathcal{X}}^{iter}$ is updated with known rows in the corrupt tensor $\widetilde{\mathcal{X}}$ (step 20), and the Forbenious norm difference between successive iterations of the completed tensor $\widehat{\mathcal{X}}$ is compared to a tolerance η to determine if the algorithm converged (steps 21-23).

IV. ADAPTIVE LINEAR TENSOR COMPLETION (ALTEC)

The algorithms presented in Section III manipulate the singular values (SiLRTC and HaLRTC) and CP factors (FCP) of an unfolded tensor to reconstruct the missing elements of the corrupt tensor. While each singular value or CP factor highlights some relevant features in the tensor, it does not capture all of it. In addition, SVD and CPD computation are expensive, and in the above-mentioned algorithms, these need to be performed at each iteration.

In this section we present a simple tensor completion method specifically designed to recover missing rows of a deep feature tensor. We refer to it as Adaptive Linear Tensor Completion (ALTeC). The proposed method assumes an approximate linear relationship among the rows of a deep feature tensor and its neighbors. Let $\mathbf{x}_i^{(c)}$ be the i-th row in channel c of tensor $\mathcal{X}$. The focus on rows comes from the specific row-by-row packetization scheme described in Section II-D. If a different packetization scheme is adopted, $\mathbf{x}_i^{(c)}$ and its neighborhood would need to be redefined, but the methodology below would still be applicable. We assume that $\mathbf{x}_i^{(c)}$ can be approximated by a linear combination of its neighbors – co-located rows in other channels and two spatial neighbors in the same channel, one above and one below:

$$\mathbf{x}_i^{(c)} \approx \sum_{j=1}^{n} w_j^{(c)} \mathbf{x}_i^{(j)} + w_{n+1}^{(c)} \mathbf{x}_{i-1}^{(c)} + w_{n+2}^{(c)} \mathbf{x}_{i+1}^{(c)}, \tag{11}$$

where $w_j^{(c)}$'s are the weights and $w_i^{(c)}$ (the weight for the row itself in the sum on the right-hand side) is set to zero. When $\mathbf{x}_i^{(c)}$ is at the top (bottom) of channel c, it's top (bottom) neighbor $\mathbf{x}_{i-1}^{(c)}$ ($\mathbf{x}_{i+1}^{(c)}$) is not available, so it is assumed to be all-zero. The above equation can be written in a matrix-vector form as

$$\mathbf{x}_i^{(c)} \approx \mathbf{X}_i^{(c)} \mathbf{w}_i^{(c)}, \tag{12}$$

where the neighbor rows of $\mathbf{x}_i^{(c)}$ have been stacked into matrix $\mathbf{X}_i^{(c)}$ as columns, and the corresponding weights have been placed into the column vector $\mathbf{w}_i^{(c)}$. Finding the optimal weights amounts to solving the following problem:

$$\min_{\mathbf{w}_i^{(c)}} \left\| \mathbf{x}_i^{(c)} - \mathbf{X}_i^{(c)} \mathbf{w}_i^{(c)} \right\|_2^2, \tag{13}$$

which has a well-known solution [15]:

$$\mathbf{w}_i^{(c)} = \left[\left(\mathbf{X}_i^{(c)} \right)^T \mathbf{X}_i^{(c)} \right]^{-1} \left(\mathbf{X}_i^{(c)} \right)^T \mathbf{x}_i^{(c)}. \tag{14}$$

For obtaining the weights $\mathbf{w}_i^{(c)}$, we used 5,000 randomly selected images from the validation set of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [16]. This training set was separate from the test set used to compare the tensor completion methods in Section V. Every input image generates a deep feature tensor at a chosen intermediate layer of a given deep model, so that $\mathbf{w}_i^{(c)}$ can be computed from (14) for every row i and every channel c. These $\mathbf{w}_i^{(c)}$'s are averaged across all training images. Further, in order to reduce weight

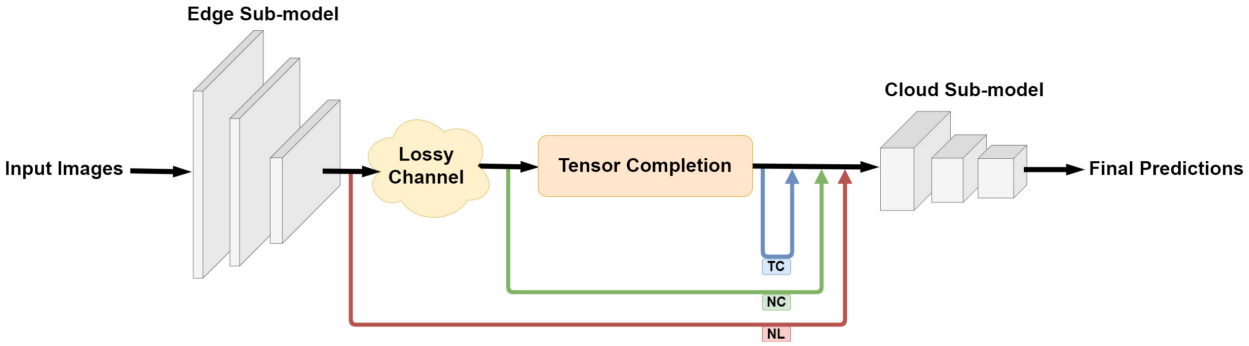

FIGURE 1. Test conditions: NL (no loss, baseline), NC (no tensor completion), TC (tensor completion).

storage requirements, we averaged all the weights for different rows i in each channel, to eventually obtain weights $\mathbf{w}^{(c)}$. The resulting weights can now be stored in a $(n+2)\times n$ matrix

$$\mathbf{W} = \left[\mathbf{w}^{(1)} \,|\, \mathbf{w}^{(2)} \,|\, \cdots \,|\, \mathbf{w}^{(n)}\right], \tag{15}$$

where the c-th column represents the weights for channel c of the deep feature tensor.

The process of tensor completion is summarized in Algorithm 4. The algorithm takes in the corrupt tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ and the weight matrix $\mathbf{W}$. First, the output tensor $\widehat{\boldsymbol{\mathcal{X}}}$ is initialized (step 1). Then the algorithm loops through the tensor channels and each missing row is reconstructed as a linear combination of its neighbors using the corresponding weights (step 7). Note that if some of the neighbors of the missing rows are also missing, the corresponding column in $\widehat{\mathbf{X}}_i^{(c)}$ is all-zero, so the corresponding term is effectively eliminated from the linear combination in (11).

As seen above, tensor reconstruction in ALTeC is linear in nature, and the linear combination for row reconstruction changes from channel to channel (hence "adaptive"). The key feature that makes ALTeC attractive compared to SiLRTC, HaLRTC, and FCP for collaborative intelligence applications, where latency is important, is its speed. ALTeC does not use computationally expensive operations such as shrinkage-based SVD or CPD. Moreover, ALTeC only reconstructs missing rows, whereas SiLRTC, HaLRTC, and FCP update the entire tensor in each iteration. Despite its relative simplicity, ALTeC achieves similar reconstruction accuracy as the other three methods, as will be seen in the next section.

V. EXPERIMENTS

A. EXPERIMENTAL SETUP

In this section, we present experiments to compare the performance of the four tensor completion algorithms described earlier – SiLRTC, HaLRTC, FCP, and ALTeC – on two pre-trained deep models for image classification, VGG16 [17] and ResNet34 [18], implemented in Keras.[2] Even though they are no longer state-of-the-art models for image classification, we selected these two models because they are well known in the research community and well-studied in the literature under a variety of application scenarios, including collaborative intelligence [1], [2]. The data used in the experiments is a randomly-selected subset of 1,000 images from the ILSVRC [16] validation set, which were different from the 5,000 images on which ALTeC was trained.

Following a common practice in video streaming simulations [10], we consider an independent random packet loss channel with packet loss probability $p_{loss} \in \{5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$. It is assumed that at the receiver (cloud sub-model), missing packets are identified via packet sequence numbers provided by a transport-layer protocol such as the Real-time Transport Protocol (RTP) [19].

For each value of p_{loss}, each image in the test set is input to the edge sub-model and the resulting deep feature tensor $\boldsymbol{\mathcal{X}}$ is transmitted over the packet loss channel $N = 100$ times, to obtain statistically meaningful results. For each of these $N = 100$ channel realizations, on the receiver side, a corrupt tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ is received, a specific tensor completion method is executed to obtain the completed tensor $\widehat{\boldsymbol{\mathcal{X}}}$, and this tensor is input to the cloud sub-model to complete the inference task. Since we focus on image classification models, we measure classification accuracy under three conditions illustrated in Fig. 1:

- no loss (NL), to establish baseline performance;
- no tensor completion (NC), where all missing data is assumed to be zero;
- tensor completion (TC), where a specific tensor completion algorithm is performed on the corrupt tensor.

The average Top-1 classification accuracy for the three cases (μ_{NL}, μ_{NC}, and μ_{TC}) and the standard deviation of Top-1 classification accuracy under NC and TC conditions (σ_{NC} and σ_{TC}) are measured. It is important to note that for each combination of packet loss level and tensor completion algorithm, random packet loss is simulated using a random seed corresponding to the trial index (1, 2, …, 100), so that each completion algorithm sees the same set of packet loss realizations. This ensures the fairness of the comparison of the four algorithms.

Before moving on to the performance comparison, we examine the convergence of SiLRTC, HaLRTC, and FCP. Recall from Section III that these algorithms iteratively

[2]https://keras.io/applications/

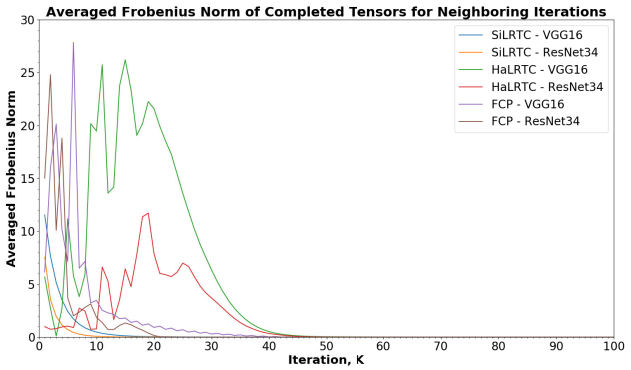

FIGURE 2. Average Frobenius norm of the differences between completed tensors in neighboring iterations.

update the to-be-completed tensor via unfolding and shrinking (or CPD) for a given number of iterations K. Fig. 2 shows the Frobenius norm ($\|\cdot\|_F$) of the difference between tensors in two consecutive iterations (K and $K+1$):

$$\left\|\widehat{\boldsymbol{\mathcal{X}}}^{K+1}-\widehat{\boldsymbol{\mathcal{X}}}^{K}\right\|_F=\|\widehat{\boldsymbol{\mathcal{A}}}\|_F=\left[\sum_{i,j,l}|\widehat{a}_{i,j,l}|^2\right]^{\frac{1}{2}} \quad (16)$$

The curves in Fig. 2 represent the average Frobenius norm of the difference between tensors in two consecutive iterations across the six packet loss probabilities (p_{loss}). As seen in the figure, all three algorithms have essentially converged and tensor updates have stopped at around $K = 50$. Hence, we use $K = 50$ iterations in our experiments for these three algorithms.

All four tensor completion algorithms were implemented and tested on a Linux based machine with the following specifications:

- Ubuntu 16.04.5 LTS (Xenial Xerus)
- 128GB (12GB) of CPU RAM (GPU RAM)
- Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz
- 12 processor cores
- Titan X (Pascal) GPU
- Tensorflow 1.9.0, Keras 2.2.4, Python 3.6.7

B. RESULTS ON VGG16

Fig. 3 shows the block diagram of the VGG16 network, and indicates the point where the network is split into the edge sub-model and the cloud sub-model. The split is at the output of the "block4_pool" layer. At this point, the feature tensor is of size $14 \times 14 \times 512$, and its total number of elements is less than the number of pixels in the input image (which is $224 \times 224 \times 3$). The volume of data to be transferred from the edge device to the cloud is an important consideration in collaborative intelligence [1], [20] and one wants to choose a split point where the data volume in the feature tensor is less than the data volume of the input. With such a split, the edge sub-model contains 7,635,264 (5.52%) of the total (trainable) parameters of the VGG16 network, and the cloud sub-model contains the remaining

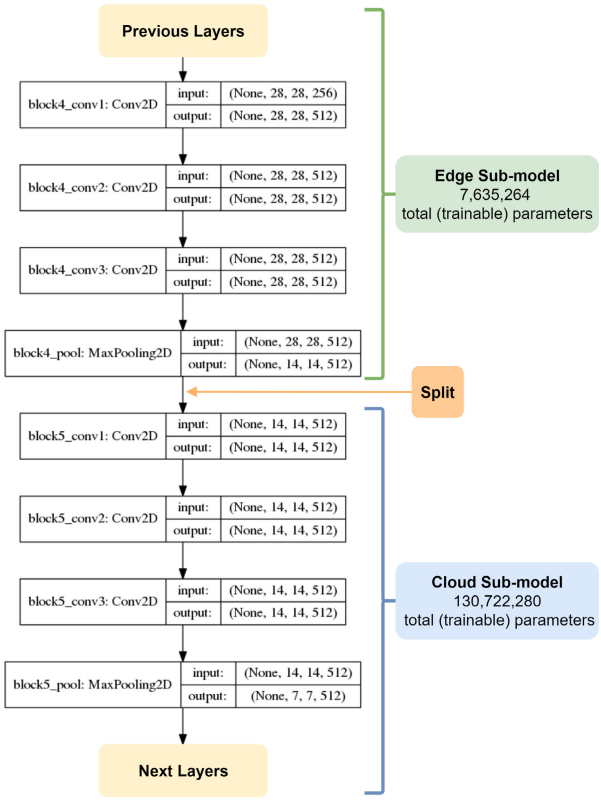

FIGURE 3. VGG16 model split at layer "block4_pool".

130,722,280 (94.48%) of the total (trainable) parameters. Again, this is a reasonable workload distribution considering the computational resources available at the edge and in the cloud.

Note that in the VGG16 network, each convolutional layer applies the Rectified Linear Unit (ReLU) activation to its output. Hence, the resulting feature tensors already contain many zero elements. Since the missing rows in the received tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ are initially filled in with zeros (the NC case), we can expect that the non-completed tensors $\widetilde{\boldsymbol{\mathcal{X}}}$ will be relatively similar to the completed tensors $\widehat{\boldsymbol{\mathcal{X}}}$. To illustrate this point, in Fig. 4 we show the percentage of zero and non-zero elements in the feature tensors for each loss value. As seen in the figure, the percentage of zero elements in the feature tensors under no loss ($p_{loss} = 0\%$) is already more than 75% and does not increase proportionally to packet loss. Hence, we expect that in this case, the difference in classification between NC and TC cases will be relatively small, as will be confirmed by quantitative results later in this section.

1) EXECUTION SPEED

First we compare the execution speed of the four tensor completion algorithms and present the results in Fig. 5. The solid curves in the figure represent the average tensor completion speed over the test set of 1,000 images, across various packet loss levels. The shaded band around solid curves indicates one standard deviation of the execution speed. As seen in

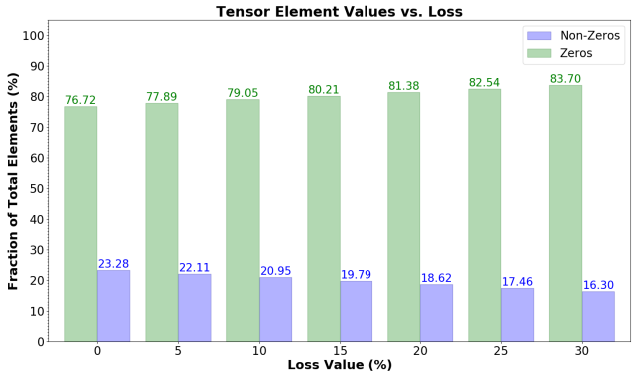

FIGURE 4. Percentage of zero elements in the VGG16 tensor as a function of packet loss.

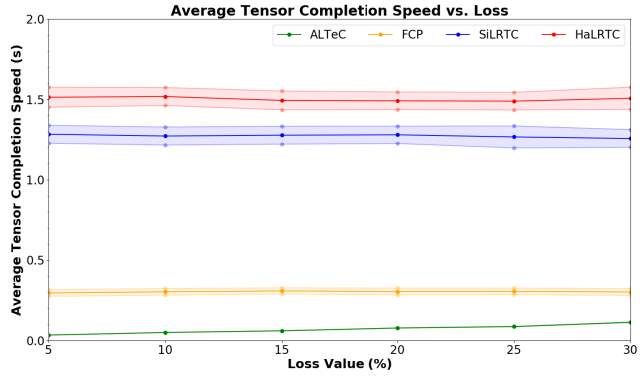

FIGURE 5. Execution speed of tensor completion algorithms - VGG16.

the figure, ALTeC is significantly faster than the other methods, with FCP being the second-fastest (due to its built-in convergence metric), followed by SiLRTC and HaLRTC, as expected.

Part of the reason for the speed advantage of ALTeC over the other three algorithms comes from avoiding the shrinkage-based SVD and CPD at each iteration. Another reason is that, in each iteration, SiLRTC, HaLRTC, and FCP update all elements of the tensor only to replace the non-missing values once the update is done. Meanwhile, ALTeC only spends computation on the tensor elements that are actually missing. ALTeC does require off-line training, however, this can be done at the same time when the main model is trained, so in terms of the overall collaborative system design, it does not add any extra overhead, and yet results in run-time savings upon system deployment.

2) CLASSIFICATION ACCURACY

Table 1 shows the Top-1 classification accuracy and its standard deviation for the several cases. The first three numerical columns show the accuracy under no packet loss (μ_{NL}), the accuracy with no tensor completion (μ_{NC}) and its standard deviation (σ_{NC}). Note that due to the large number of zeros produced by the ReLU activation functions in certain VGG16 layers (Fig. 4), μ_{NC} is fairly close to μ_{NL},

TABLE 1. VGG16 classification accuracy results. Among the last four columns, the first two show the results with default settings, and the last two show the results with matched execution speed.

					Default		Speed-matched	
p_{loss}	Algorithm	μ_{NL}	μ_{NC}	σ_{NC}	μ_{TC}	σ_{TC}	μ_{TC}	σ_{TC}
5%	SiLRTC	56.20%	55.96%	0.41%	56.09%	0.39%	55.96%	0.41%
	HaLRTC	56.20%	55.96%	0.41%	56.06%	0.36%	55.96%	0.41%
	FCP	56.20%	55.96%	0.41%	56.09%	0.41%	55.99%	0.44%
	ALTeC	56.20%	55.96%	0.41%	56.07%	0.40%	56.07%	0.40%
10%	SiLRTC	56.20%	55.50%	0.55%	55.67%	0.44%	55.53%	0.52%
	HaLRTC	56.20%	55.50%	0.55%	55.75%	0.37%	55.51%	0.54%
	FCP	56.20%	55.50%	0.55%	55.78%	0.51%	55.78%	0.53%
	ALTeC	56.20%	55.50%	0.55%	55.70%	0.48%	55.70%	0.48%
15%	SiLRTC	56.20%	54.76%	0.60%	54.99%	0.58%	54.79%	0.62%
	HaLRTC	56.20%	54.76%	0.60%	55.14%	0.44%	54.75%	0.59%
	FCP	56.20%	54.76%	0.60%	55.17%	0.55%	55.15%	0.59%
	ALTeC	56.20%	54.76%	0.60%	55.11%	0.56%	55.11%	0.56%
20%	SiLRTC	56.20%	54.18%	0.63%	54.51%	0.61%	54.24%	0.64%
	HaLRTC	56.20%	54.18%	0.63%	54.67%	0.55%	54.21%	0.64%
	FCP	56.20%	54.18%	0.63%	54.74%	0.65%	54.72%	0.67%
	ALTeC	56.20%	54.18%	0.63%	54.64%	0.63%	54.64%	0.63%
25%	SiLRTC	56.20%	53.45%	0.79%	53.95%	0.69%	53.51%	0.76%
	HaLRTC	56.20%	53.45%	0.79%	54.19%	0.67%	53.48%	0.80%
	FCP	56.20%	53.45%	0.79%	54.16%	0.71%	54.16%	0.72%
	ALTeC	56.20%	53.45%	0.79%	54.03%	0.75%	54.03%	0.75%
30%	SiLRTC	56.20%	52.57%	0.77%	53.13%	0.73%	52.69%	0.78%
	HaLRTC	56.20%	52.57%	0.77%	53.39%	0.67%	52.65%	0.78%
	FCP	56.20%	52.57%	0.77%	53.31%	0.78%	53.26%	0.74%
	ALTeC	56.20%	52.57%	0.77%	53.25%	0.81%	53.25%	0.81%

as predicted earlier. Even under 30% loss, the classification accuracy drops by less than 4%.

The next two columns show the accuracy (μ_{TC}) and standard deviation (σ_{TC}) for the four tensor completion algorithms with default settings. By this we mean that SiLRTC, HaLRTC, and FCP are run for $K = 50$ iterations, as explained earlier. However, this means that their execution speed is significantly higher than that of ALTeC (Fig. 5). Therefore, in the last two columns, we report the results for the matched execution speed, where we only let SiLRTC, HaLRTC, and FCP run until their execution matches that of ALTeC. This means that they were only able to run 1-5 (SiLRTC & HaLRTC) or 5-20 (FCP) iterations, depending on the case.

In the default-settings case, the accuracies of all four methods were similar, with the maximum difference of around 0.25% (between SiLRTC and HaLRTC) under 30% loss. To test for statistical significance of these differences, we applied Welch's t-test for samples with unequal variance [21]. The resulting p-values are shown in the middle three columns of Table 2. In experimental sciences, a p-value of less than 0.05 is usually taken as a sign of statistically significant difference. As seen in Table 2, all pairwise

TABLE 2. VGG16 statistical significance results. The middle three columns show *p*-values of the 2-sided Welch's t-test for pairwise comparison of accuracies with default settings, while the last three columns show *p*-values for the case when the execution speeds are matched.

		Default			Speed-matched		
p_{loss}	Algorithm	ALTeC	SiLRTC	HaLRTC	ALTeC	SiLRTC	HaLRTC
5%	SiLRTC	0.669	-	-	0.054	-	-
	HaLRTC	0.897	0.563	-	0.053	0.986	-
	FCP	0.662	0.986	0.558	0.199	0.567	0.575
10%	SiLRTC	0.681	-	-	0.020	-	-
	HaLRTC	0.435	0.199	-	0.009	0.721	-
	FCP	0.245	0.108	0.592	0.268	0.001	4.351E-04
15%	SiLRTC	0.149	-	-	1.423E-04	-	-
	HaLRTC	0.666	0.059	-	1.563E-05	0.658	-
	FCP	0.419	0.025	0.700	0.148	5.519E-07	3.522E-08
20%	SiLRTC	0.132	-	-	1.083E-05	-	-
	HaLRTC	0.740	0.052	-	2.675E-06	0.733	-
	FCP	0.272	0.009	0.400	0.421	5.590E-07	1.282E-07
25%	SiLRTC	0.461	-	-	3.272E-06	-	-
	HaLRTC	0.110	0.015	-	1.454E-06	0.773	-
	FCP	0.201	0.037	0.769	0.201	3.636E-09	1.672E-09
30%	SiLRTC	0.264	-	-	1.242E-06	-	-
	HaLRTC	0.183	0.009	-	2.640E-07	0.731	-
	FCP	0.626	0.099	0.404	0.942	2.968E-07	5.526E-08

differences between average accuracies in the default-settings case were insignificant, except for the difference between SiLRTC & HaLRTC for 25% & 30% loss and SiLRTC & FCP for 15%, 20%, and 25% loss, and these cases are indicated with green shading in the table. Since no algorithm came out as the clear winner (i.e., provided significantly better results than *all* alternatives) no accuracy in the corresponding column in Table 1 is indicated in bold.

Likewise, in the matched-speed case, there was no clear winner that statistically outperformed *all* its rivals, as indicated by *p*-values in Table 2. Thus, again, no accuracy in the corresponding column in Table 1 is indicated in bold. Overall, results obtained from the VGG16 model suggest that ALTeC offers equivalent performance to SiLRTC, HaLRTC, and FCP, both when complexity is not constrained and when the algorithms are constrained to be equally fast.

3) INTERESTING EXAMPLES

Finally, we highlight several interesting examples that were observed during testing of tensor completion algorithms on VGG16. Here, all tensor completion algorithms run in the default (i.e., not speed-matched) configuration. Table 3 shows classification predictions made on two images: #102 ('Sleeping Bag' or 'SB') and #3 ('Bulbul'). Ground truth labels are listed in the row that starts with ''GT Label''. The next two rows show the results under no loss (NL). In other words, these are the outputs obtained from the pre-trained VGG16 model. For image #102, we see that the Top-1 label is wrong ('Cloak') and the model is fairly confident about it (74.83%). After packet loss of 30%, but without tensor completion (NC), the model is still wrong ('Cloak'), but it is less confident than before (49.73%). Finally, after tensor completion (TC), ALTeC produces the correct result ('SB'),

TABLE 3. Interesting examples from VGG16 experiments.

Image	102				3			
Trial #	85				35			
Algorithm	ALTeC	FCP	HaLRTC	SiLRTC	ALTeC	FCP	HaLRTC	SiLRTC
GT Label	SB	SB	SB	SB	Bulbul	Bulbul	Bulbul	Bulbul
NL Label	Cloak	Cloak	Cloak	Cloak	Bulbul	Bulbul	Bulbul	Bulbul
NL Score	74.83%	74.83%	74.83%	74.83%	95.80%	95.80%	95.80%	95.80%
NC Label	Cloak	Cloak	Cloak	Cloak	Kite	Kite	Kite	Kite
NC Score	49.73%	49.73%	49.73%	49.73%	42.24%	42.24%	42.24%	42.24%
TC Label	SB	Cloak	Cloak	Cloak	Kite	Kite	Kite	Kite
TC Score	51.67%	51.85%	55.24%	56.37%	**36.93%**	43.13%	45.45%	46.02%

SB = Sleeping Bag

while SiLRTC, HaLRTC, and FCP are wrong, but not as confident about it as the original model (around 52-56%).

For image #3, the pre-trained model is correct ('Bulbul'), and also confident about it (95.80%). After packet loss, but without tensor completion (NC), the model produces wrong result ('Kite') with confidence of 42.24%. All tensor completion algorithms lead to wrong result ('Kite'), but ALTeC leads to least confidence about this result (36.93%) while SiLRTC, HaLRTC, and FCP increase their confidence about the wrong decision (to 43-46%) compared to the NC case. Although all tensor completion methods lead to wrong decision in this case, one could argue that ALTeC is still better than the other three methods because with ALTeC-completed tensor, the VGG16 model is least confident about its wrong answer.

C. RESULTS ON RESNET34

Fig. 6 shows the architecture of the ResNet34 model. By similar reasoning as in Section V-B, we decided to split the

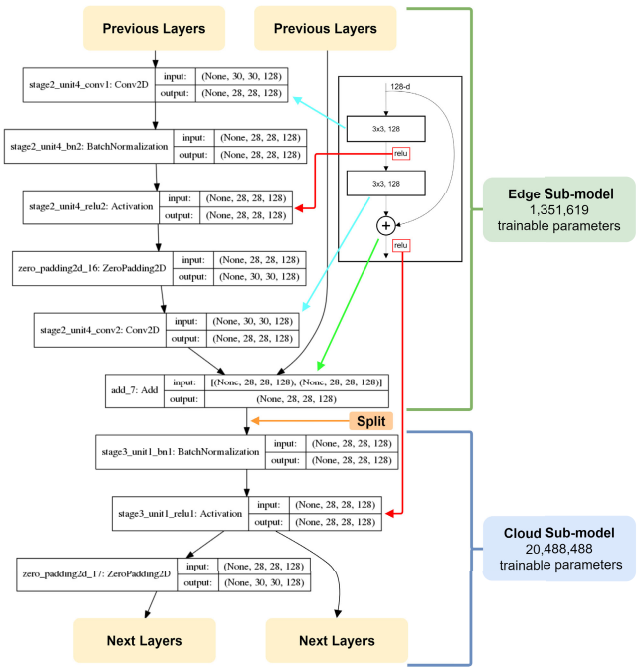

FIGURE 6. ResNet34 model split at layer "add_7" [18].

model at layer "add_7". With this split, the edge sub-model produces tensors of size 28 × 28 × 128, which contain the same number of elements as in the VGG16 case. Also, the edge sub-model contains 1,351,619 (6.19%) trainable parameters, while the cloud sub-model contains the remaining 20,488,488 (93.81%) trainable parameters. Again, this is a reasonable workload distribution considering the computational resources available at the edge and in the cloud.

ResNet34, like VGG16, uses ReLU activation functions. However, it also makes use of batch normalization [22], which centers the data distribution prior to adding it to the data passed down the residual (skip) connection [18]. Hence, the tensor produced by the edge sub-model in this case does not contain as many zeros as the one produced by the VGG16 edge sub-model. Indeed, Fig. 7 shows that without loss, there are virtually no zeros in the tensor $\mathcal{X}$ produced by the ResNet34 edge sub-model. As the loss increases, the percentage of zero-elements in the received tensor $\widetilde{\mathcal{X}}$ increases proportionally, and we can expect a larger difference between non-completed tensors $\widetilde{\mathcal{X}}$ and completed tensors $\widehat{\mathcal{X}}$ than we had in the VGG16 case.

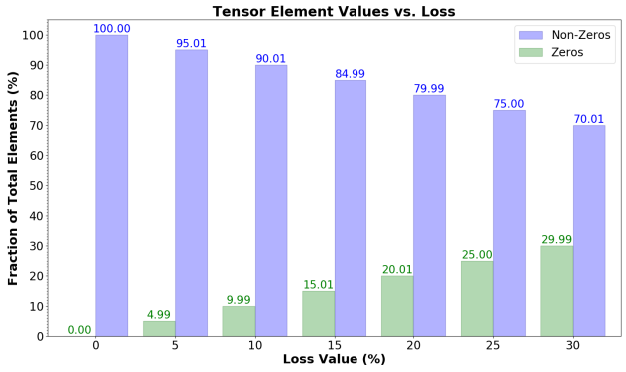

FIGURE 7. Percentage of zero elements in the ResNet34 tensor as a function of packet loss.

1) EXECUTION SPEED

Similarly to the VGG16 case, we tested the execution speed of the four tensor completion algorithms and report the results in Fig. 8. The solid curves represent the average tensor completion speed over the test set of 1,000 images, across various packet loss levels. The shaded band around solid curves indicates one standard deviation of the execution speed. As before, ALTeC is significantly faster than the other three methods, with FCP being the second-fastest, followed by SiLRTC and HaLRTC.

When comparing these results to those in Fig. 5, we note that the execution speeds of SiLRTC and HaLRTC are similar, about 1.2–1.5 seconds/tensor. Likewise, the execution speeds of FCP are similar at about 0.25–0.30 seconds/tensor. This is not surprising considering that in both VGG16 and ResNet34 cases, the tensors (and their unfolded versions) have the same number of elements. However, ALTeC is

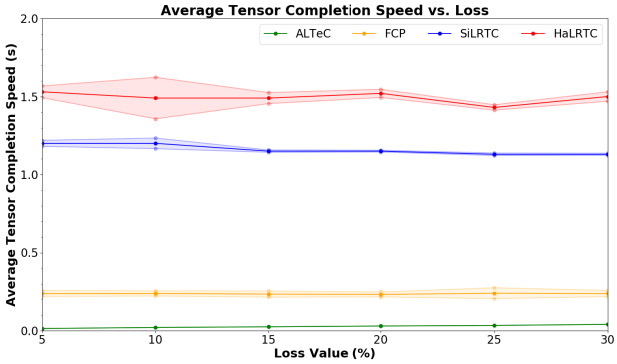

FIGURE 8. Execution speed of tensor completion algorithms - ResNet34.

TABLE 4. ResNet34 classification accuracy results. Among the last four columns, the first two show the results with default settings, and the last two show the results with matched execution speed.

					Default		Speed-matched	
p_{loss}	Algorithm	μ_{NL}	μ_{NC}	σ_{NC}	μ_{TC}	σ_{TC}	μ_{TC}	σ_{TC}
5%	SiLRTC	58.10%	57.57%	0.61%	57.77%	0.49%	57.75%	0.53%
	HaLRTC	58.10%	57.57%	0.61%	57.94%	0.37%	57.75%	0.61%
	FCP	58.10%	57.57%	0.61%	57.92%	0.43%	57.59%	0.60%
	ALTeC	58.10%	57.57%	0.61%	58.04%	0.44%	**58.04%**	0.44%
10%	SiLRTC	58.10%	54.57%	0.68%	56.47%	0.60%	56.12%	0.66%
	HaLRTC	58.10%	54.57%	0.68%	**57.65%**	0.46%	54.57%	0.68%
	FCP	58.10%	54.57%	0.68%	56.56%	0.66%	55.98%	0.69%
	ALTeC	58.10%	54.57%	0.68%	57.18%	0.61%	**57.18%**	0.61%
15%	SiLRTC	58.10%	49.30%	0.78%	53.89%	0.64%	52.84%	0.71%
	HaLRTC	58.10%	49.30%	0.78%	**57.02%**	0.51%	49.31%	0.78%
	FCP	58.10%	49.30%	0.78%	53.96%	0.75%	53.20%	0.78%
	ALTeC	58.10%	49.30%	0.78%	55.09%	0.71%	**55.09%**	0.71%
20%	SiLRTC	58.10%	40.87%	0.86%	49.61%	0.77%	48.64%	0.80%
	HaLRTC	58.10%	40.87%	0.86%	**56.26%**	0.60%	40.87%	0.86%
	FCP	58.10%	40.87%	0.86%	49.76%	0.76%	49.10%	0.87%
	ALTeC	58.10%	40.87%	0.86%	51.99%	0.72%	**51.99%**	0.72%
25%	SiLRTC	58.10%	29.11%	0.86%	43.56%	0.87%	41.40%	0.99%
	HaLRTC	58.10%	29.11%	0.86%	**55.09%**	0.65%	29.11%	0.87%
	FCP	58.10%	29.11%	0.86%	44.10%	0.81%	43.07%	0.82%
	ALTeC	58.10%	29.11%	0.86%	47.52%	0.67%	**47.52%**	0.67%
30%	SiLRTC	58.10%	15.72%	0.77%	34.56%	0.89%	31.85%	0.83%
	HaLRTC	58.10%	15.72%	0.77%	**53.63%**	0.68%	15.73%	0.77%
	FCP	58.10%	15.72%	0.77%	36.06%	0.76%	34.93%	0.80%
	ALTeC	58.10%	15.72%	0.77%	41.23%	0.80%	**41.23%**	0.80%

noticeably faster on ResNet34 tensors (Fig. 8) than on VGG16 tensors (Fig. 5), because ResNet34 tensors have fewer channels than VGG16 tensors, so matrix $\widehat{\mathbf{X}}_i^{(c)}$ in step 7 of ALTeC is smaller in the ResNet34 case, which leads to faster matrix-vector multiplication.

2) CLASSIFICATION ACCURACY

As in the VGG16 case, we compute the classification accuracy offered by the four tensor completion methods in the default case (where SiLRTC, HaLRTC, and FCP run $K = 50$ iterations) and in the speed-matched case, where they are only allowed to run as long as ALTeC. The results are shown in Table 4. First, note that in the case of ResNet34, there

is a large difference between the no-loss case (NL) and the the case with no tensor completion (NC). Specifically, for 30% loss, the difference in Top-1 classification accuracy is now over 40%, whereas in the case of VGG16 it was less than 4%. As mentioned earlier, this is due to the fact that tensors $\mathcal{X}$ produced by the ResNet34 edge sub-model (without loss) contain virtually no zeros, so they are quite different compared to the corrupt $\widetilde{\mathcal{X}}$ which haven't been completed yet. This also means that tensor completion (TC) has the potential to bring much higher gain over no-completion (NC), compared to the VGG16 case.

In Table 4, the middle two columns labeled μ_{TC} and σ_{TC} correspond to the case with default settings, where SiLRTC, HaLRTC, and FCP are able to execute all $K = 50$ iterations until convergence. The corresponding statistical significance results using Welch's t-test are shown in the middle three columns in Table 5. As seen in Table 5, we have a lot more significant differences in accuracy now, compared to the VGG16 case. At the 5% loss level, ALTeC, HaLRTC, and FCP are statistically tied, and all give a higher classification accuracy than SiLRTC. At higher loss levels, HaLRTC statistically outperforms ALTeC, SiLRTC, and FCP, with ALTeC being the next best performing algorithm, followed by FCP and SiLRTC. For this reason, the corresponding accuracies are made bold in the table.

TABLE 5. ResNet34 statistical significance results. The middle three columns show *p*-values of the 2-sided Welch's t-test for pairwise comparison of accuracies with default settings, while the last three columns show *p*-values for the case when the execution speeds are matched.

		Default			Speed-matched		
p_{loss}	Algorithm	ALTeC	SiLRTC	HaLRTC	ALTeC	SiLRTC	HaLRTC
5%	SiLRTC	7.350E-06	-	-	4.429E-05	-	-
	HaLRTC	0.105	0.005	-	3.930E-09	0.031	-
	FCP	0.059	0.022	0.687	1.198E-08	0.055	0.806
10%	SiLRTC	3.795E-15	-	-	2.422E-24	-	-
	HaLRTC	2.840E-09	2.646E-35	-	3.336E-71	1.982E-38	-
	FCP	1.491E-10	0.317	7.694E-29	6.210E-28	0.148	6.408E-33
15%	SiLRTC	1.159E-26	-	-	6.368E-56	-	-
	HaLRTC	9.902E-53	2.881E-90	-	1.673E-120	8.258E-83	-
	FCP	7.900E-28	0.178	3.503E-82	8.240E-26	1.285E-16	6.624E-98
20%	SiLRTC	3.238E-56	-	-	3.136E-77	-	-
	HaLRTC	2.197E-104	2.870E-133	-	1.056E-166	3.435E-136	-
	FCP	1.405E-52	0.165	2.225E-132	5.998E-47	1.654E-18	6.802E-145
25%	SiLRTC	7.169E-86	-	-	3.544E-106	-	-
	HaLRTC	5.468E-153	8.973E-167	-	5.335E-205	7.427E-163	-
	FCP	8.255E-80	8.591E-06	2.027E-170	9.173E-65	6.254E-64	2.852E-193
30%	SiLRTC	4.285E-122	-	-	2.051E-153	-	-
	HaLRTC	4.935E-182	1.625E-205	-	7.433E-242	9.334E-200	-
	FCP	7.096E-109	1.140E-27	2.085E-215	7.789E-88	8.239E-108	4.573E-227

The columns labelled "Speed-matched" in both Table 4 and Table 5 correspond to the case when the execution speeds of the four tensor completion algorithms are matched. In this case, SiLRTC and HaLRTC are only able to run 1-3 iterations, while FCP can run 3-8 iterations. Under these conditions, ALTeC statistically outperforms the other three methods at all loss levels. In summary, ResNet34 results show that when complexity is not constrained and execution speed is of no concern, HaLRTC is the best of the four methods, followed by ALTeC in the second place. When the execution speeds are matched, ALTeC is superior to the other three methods.

TABLE 6. Interesting examples from ResNet34 experiments.

Image	4				7			
Trial #	52				27			
Algorithm	ALTeC	FCP	HaLRTC	SiLRTC	ALTeC	FCP	HaLRTC	SiLRTC
GT Label	Toyshop	Toyshop	Toyshop	Toyshop	Drake	Drake	Drake	Drake
NL Label	Toyshop	Toyshop	Toyshop	Toyshop	Drake	Drake	Drake	Drake
NL Score	16.66%	16.66%	16.66%	16.66%	78.19%	78.19%	78.19%	78.19%
NC Label	Abacus	Abacus	Abacus	Abacus	Ptarmigan	Ptarmigan	Ptarmigan	Ptarmigan
NC Score	7.89%	7.89%	7.89%	7.89%	26.57%	26.57%	26.57%	26.57%
TC Label	**Toyshop**	WS	CR	SC	AC	AC	AC	AC
TC Score	6.32%	8.47%	20.06%	6.89%	**57.92%**	59.83%	75.67%	62.62%

WS = Window Screen, CR = Coral Reef, SC = Shower Curtain, AC = American Coot

3) INTERESTING EXAMPLES

Again, we highlight several interesting examples that were observed during the experiments on ResNet34 with default configurations of the tensor completion algorithms. Table 6 shows classification predictions made on two images: #4 ('Toyshop') and #7 ('Drake'). Ground truth labels are listed in the row that starts with "GT Label". The next two rows show the results under no loss (NL), obtained from the pre-trained ResNet34 model. In both cases, Top-1 labels are correct, but the model is relatively less confident about image #4 (16.66%). After packet loss of 30%, but without tensor completion (NC), the model makes wrong predictions in both cases - image #4 is classified as 'Abacus' and #7 is classified as 'Ptarmigan' - though both predictions are made with low confidence (7.89% and 26.57%, respectively).

Finally, after tensor completion (TC), ALTeC produces correct result in the case of image #4 ('Toyshop'), but with low confidence (6.32%), while SiLRTC, HaLRTC, and FCP are wrong. In the case of image #7, all four methods produce wrong results, but ALTeC is the least confident about it (57.92% vs. 60-75% for other methods). Again, among the four wrong decisions, the best one is where the model is least confident about it. Note that in the case of image #7, although final TC labels are wrong for all the methods, they are relatively similar to the ground truth, as both 'Drake' and 'American Coot' are birds that resemble each another.

VI. CONCLUSION AND FUTURE WORK

In this paper, we studied several methods for tensor completion in collaborative intelligence applications. Specifically, we focused on three representative methods from the literature - Simple Low Rank Tensor Completion (SiLRTC), High-accuracy Low Rank Tensor Completion (HaLRTC), and Fused Canonical Polyadic (FCP) - and a simple newly-developed Adaptive Linear Tensor Completion (ALTeC). These methods were compared on their ability to recover the missing data caused by packet loss in the feature tensors produced by VGG16 and ResNet34 image classification models.

Algorithm 1 SiLRTC [5]

Input: $\widetilde{\mathcal{X}}, K, \alpha_i, \beta_i \geq 0, \sum_{i=1}^{3} \alpha_i = 1$
Output: $\widehat{\mathcal{X}}$
$\widehat{\mathcal{X}} \leftarrow \widetilde{\mathcal{X}}$
for $iter = 1, 2, \ldots, K$, **do**
  $\mathcal{M} \leftarrow \text{zeros}(\widehat{\mathcal{X}}.shape)$
  for $i = 1, 2, 3$, **do**
    $\mathbf{X} \leftarrow \text{unfold}(\widehat{\mathcal{X}}, i)$
    $\mathbf{X}_\tau \leftarrow \text{shrink}(\mathbf{X}, \ \tau = \alpha_i/\beta_i)$
    $\mathcal{X}_\tau \leftarrow \text{fold}(\beta_i \cdot \mathbf{X}_\tau, i)$
    $\mathcal{M} \leftarrow \mathcal{M} + \mathcal{X}_\tau$
  end for
  $\mathcal{M} \leftarrow \mathcal{M} / \sum_{i=1}^{3} \beta_i$
  Replace values in $\mathcal{M}$ by known rows in $\widetilde{\mathcal{X}}$
  $\widehat{\mathcal{X}} \leftarrow \mathcal{M}$
end for

Algorithm 2 HaLRTC [5]

Input: $\widetilde{\mathcal{X}}, K, \alpha_i, \rho \geq 0, \sum_{i=1}^{3} \alpha_i = 1$
Output: $\widehat{\mathcal{X}}$
$\widehat{\mathcal{X}} \leftarrow \widetilde{\mathcal{X}}$
for $i = 1, 2, 3$, **do**
  $\mathcal{M}_i, \mathcal{Y}_i \leftarrow \text{zeros}(\widehat{\mathcal{X}}.shape)$
end for
for $iter = 1, 2, \ldots, K$, **do**
  for $i = 1, 2, 3$, **do**
    $\mathbf{X} \leftarrow \text{unfold}(\widehat{\mathcal{X}}, i)$
    $\mathbf{Y} \leftarrow \text{unfold}(\mathcal{Y}_i, i)/\rho$
    $\mathbf{Z}_\tau \leftarrow \text{shrink}(\mathbf{X} + \mathbf{Y}, \ \tau = \alpha_i/\rho)$
    $\mathcal{M}_i \leftarrow \text{fold}(\mathbf{Z}_\tau, i)$
  end for
  $\widehat{\mathcal{X}} \leftarrow \frac{1}{3} \cdot \sum_{i=1}^{3} \left(\mathcal{M}_i - \frac{1}{\rho} \cdot \mathcal{Y}_i \right)$
  Replace values in $\widehat{\mathcal{X}}$ by known rows in $\widetilde{\mathcal{X}}$
  for $i = 1, 2, 3$, **do**
    $\mathcal{Y}_i \leftarrow \mathcal{Y}_i - \rho \cdot (\mathcal{M}_i - \widehat{\mathcal{X}})$
  end for
  $\rho \leftarrow 1.2 \cdot \rho$
end for

Among the four studied methods, ALTeC was the fastest, which is well-suited for collaborative intelligence applications where inference latency is one of the important issues. Regarding reconstruction accuracy, on VGG16 tensors (which tend to be sparse), all four methods were in a statistical tie, both when SiLRTC, HaLRTC, and FCP were allowed to run sufficiently many iterations to converge and when their execution speeds were matched with that of ALTeC by restricting the number of iterations. On ResNet34 tensors (which are non-sparse), HaLRTC showed the best accuracy when it was allowed to converge, followed by ALTeC as the second-best. However, when the execution speeds were matched, ALTeC emerged as the winner.

Algorithm 3 FCP [6]

Input: $\widetilde{\mathcal{X}}, \mathbf{A}_i, \mathbf{L}_p, \mathbf{L}_s, K, \mu, \alpha_i, \beta_i, \delta_i, \zeta_i, R, R_u, R_m, \eta, \epsilon$
Output: $\widehat{\mathcal{X}}$
$\widehat{\mathcal{X}} \leftarrow \widetilde{\mathcal{X}}$
for $iter = 1, 2, \ldots, K$, **do**
  for $i = 1, 2, 3$, **do**
    $\mathbf{S}, \mathbf{V} \leftarrow$ Compute $\mathbf{S}$ as in (9), unfold$(\widehat{\mathcal{X}}, i)$
    $L \leftarrow \left\| \mathbf{S}^T\mathbf{S} + \delta_i \mathbf{I} \right\|_F + \beta_i \left\| \mathbf{L}_p^i \right\|_F + \frac{\alpha_i}{\mu} + \frac{\zeta_i}{\mu} \left\| \mathbf{L}_s^i \right\|_F^2$
    $\nabla h(\mathbf{A}_i) \leftarrow \mathbf{S}^T\mathbf{S}\mathbf{A}_i - \mathbf{S}^T\mathbf{V} + \frac{\alpha_i}{\mu} Q(\frac{\mu}{\alpha_i}\mathbf{A}_i) + \beta_i \mathbf{A}_i (\mathbf{L}_p^i)^T + \delta_i \mathbf{A}_i + \frac{\zeta_i}{\mu} Q(\frac{1}{\mu}\mathbf{A}_i(\mathbf{L}_s^i)^T)\mathbf{L}_s^i$
    $\mathbf{Y}_0, d_0, k \leftarrow \mathbf{A}_i, 1, 0$
    while stopping_criterion $(\nabla h(\mathbf{Y}_k)) > \epsilon$ **do**
      $\mathbf{A}_i^k \leftarrow \mathbf{Y}_k - \frac{1}{L}\nabla h(\mathbf{Y}_k)$
      $d_{k+1} \leftarrow \frac{1}{2}(1 + \sqrt{4d_k^2 + 1})$
      $\mathbf{Y}_{k+1} \leftarrow \mathbf{A}_i^k + \frac{d_{k-1}}{d_{k+1}}(\mathbf{A}_i^k - \mathbf{A}_i^{k-1})$
      $k \leftarrow k + 1$
    end while
  end for
  if $\sum_{n=1}^{3} \frac{\left\| \mathbf{A}_n^{iter} - \mathbf{A}_n^{iter-1} \right\|_F}{\left\| \mathbf{A}_n^{iter-1} \right\|_F} < \eta$ **and** $R < R_m$ **then**
    $R \leftarrow R + R_u$
  end if
  $\widehat{\mathcal{X}}^{iter-1} \leftarrow \text{fold}(\mathbf{A}_1(\mathbf{A}_3 \odot \mathbf{A}_2), 1)$
  $\widehat{\mathcal{X}}^{iter} \leftarrow \widehat{\mathcal{X}}^{iter-1}$
  Replace values in $\widehat{\mathcal{X}}^{iter}$ by known rows in $\widetilde{\mathcal{X}}$
  if $\left\| \widehat{\mathcal{X}}^{iter} - \widehat{\mathcal{X}}^{iter-1} \right\|_F < \eta$ **then**
    break
  end if
end for

In essence, SiLRTC, HaLRTC, and FCP pay the price for their generality. By not embedding the specifics of tensors they are supposed to complete into their procedures, they need to re-discover low-rank tensor structures anew every time they start the completion procedure. By contrast, ALTeC learns simple linear relations among the rows of tensors it is supposed to complete off-line, so it is able to execute quickly at run-time. While ALTeC requires off-line training, this is quite feasible in collaborative intelligence because the main model (VGG16 or ResNet34 in this case) also requires off-line training, and ALTeC could be trained in parallel with the main model on the same data. Further, note that ALTeC does not require labeled data for training, only input data. Each input sample produces the "ground truth" tensor at a given layer of the model, which is then used to fit the parameters of ALTeC.

Note that both fast methods, as well as slower but more accurate methods, may have their place in CI tensor completion, depending on the application. For example, in a video surveillance application where a subway station is being monitored by several cameras to detect abandoned luggage, speed is of the essence, since the luggage may pose a security threat. Meanwhile, the accuracy of luggage classification

Algorithm 4 ALTeC

Input: $\widetilde{\mathcal{X}}$, **W**
Output: $\widehat{\mathcal{X}}$
$\widehat{\mathcal{X}} \leftarrow \widetilde{\mathcal{X}}$
for channel $c = 1, 2, \ldots, n$ **do**
  for each row i in channel c of $\widehat{\mathcal{X}}$ **do**
    if row i is missing **then**
      Collect neighbors of the i-th row into $\widehat{\mathbf{X}}_i^{(c)}$
      Extract the c-th column of **W** into $\mathbf{w}^{(c)}$
      $\widehat{\mathbf{x}}_i^{(c)} = \widehat{\mathbf{X}}_i^{(c)} \mathbf{w}^{(c)}$
      Put $\widehat{\mathbf{x}}_i^{(c)}$ into the i-th row of channel c of $\widehat{\mathcal{X}}$
    end if
  end for
end for

(suitcase, backpack, purse) is less relevant. Hence, for such an application, inference latency (i.e., speed) would be more important than classification accuracy. On the other hand, for applications such as satellite-based surveillance of crops, speed is not as important because changes on crop fields are relatively slow. Hence, in this application, accuracy could be preferred over speed.

In the future, we plan to study other tensor completion methods [23] in the context of collaborative intelligence, such as Geometric Conjugate Gradients (GeomCG) [24], Tensor SVD (t-SVD) [25], and Tensor Robust Principal Component Analysis (TRPCA) [26]. Like SiLRTC, HaLRTC, and FCP, most of these completion algorithms rely on computation-heavy procedures such as eigen-decomposition or SVD, so they are unlikely to be faster than ALTeC, but they may offer better accuracy in cases where complexity is less important. The inclusion of specific constrains found in image or video completion algorithms [27], [28] could also be explored. Furthermore, similar studies could be performed on models trained for other collaborative intelligence applications, such as object detection, segmentation, action recognition, etc. Finally, the completion algorithms should also be evaluated on a burst-loss channel model such as the Gilbert-Elliott model [29], which offers a more realistic representation of packet loss in real networks.

APPENDIX
TENSOR COMPLETION ALGORITHMS PSEUDO CODE

See Algorithms 1–4.

REFERENCES

[1] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017.

[2] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, early access, 2019, doi: 10.1109/TMC.2019.2947893.

[3] H. Choi and I. V. Bajic, "Deep feature compression for collaborative object detection," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 3743–3747.

[4] S. R. Alvar and I. V. Bajic, "Multi-task learning with compressible features for collaborative intelligence," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 1705–1709.

[5] 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.

[6] Y. Wu, H. Tan, Y. Li, J. Zhang, and X. Chen, "A fused CP factorization method for incomplete tensors," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 751–764, Mar. 2019.

[7] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Linear Algebra*. Berlin, Germany: Springer, 1971, pp. 134–151.

[8] J.-F. Cai, E. J. Candés, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM J. Optim.*, vol. 20, no. 4, pp. 1956–1982, Jan. 2010.

[9] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Error preserving correction: A method for CP decomposition at a target error bound," *IEEE Trans. Signal Process.*, vol. 67, no. 5, pp. 1175–1190, Mar. 2019.

[10] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing and Communications*, Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[11] Q. Song, H. Ge, J. Caverlee, and X. Hu, "Tensor completion algorithms in big data analytics," *ACM Trans. Knowl. Discovery Data (TKDD)*, vol. 13, no. 1, pp. 1–48, Jan. 2019.

[12] D. Kressner, M. Steinlechner, and B. Vandereycken, "Low-rank tensor completion by Riemannian optimization," *BIT Numer. Math.*, vol. 54, no. 2, pp. 447–468, Nov. 2013.

[13] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[14] J. Eckstein and D. P. Bertsekas, "On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Program.*, vol. 55, nos. 1–3, pp. 293–318, Apr. 1992.

[15] H. Stark and J. W. Woods, *Probability, Statistics, and Random Processes for Engineers*, 4th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2012.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Apr. 2015.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, San Diego, CA, USA, May 2015, pp. 1–14.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document RFC 3550, Jul. 2003.

[20] H. Choi and I. V. Bajic, "Near-lossless deep feature compression for collaborative intelligence," in *Proc. IEEE 20th Int. Workshop Multimedia Signal Process. (MMSP)*, Aug. 2018, pp. 1–6.

[21] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 2nd ed. London, U.K.: Chapman & Hall, 2000.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015, pp. 1–11.

[23] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, Aug. 2013.

[24] H. Kasai and B. Mishra, "Low-rank tensor completion: A Riemannian manifold preconditioning approach," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1012–1021.

[25] 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.

[26] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, "Tensor robust principal component analysis with a new tensor nuclear norm," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published.

[27] 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.

[28] 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.

[29] G. Hasslinger and O. Hohlfeld, "The Gilbert-Elliott model for packet loss in real time services on the Internet," in *Proc. Meas., Modeling Eval. Comput. Commun. Syst.*, Mar. 2008, pp. 1–15.

LIOR BRAGILEVSKY (Student Member, IEEE) received the B.A.Sc. degree (Hons.) in electronics engineering from Simon Fraser University, Burnaby, BC, Canada, in 2018, where he is currently pursuing the M.A.Sc. degree (engineering - deep learning), under the Supervision of Professor I. V. Bajić. Since 2017, he has been a Research Assistant at the Simon Fraser University's Multimedia Lab. During this time, he worked on a variety of machine learning projects, ranging from classification of deforestation patterns in the Amazon to developing tensor completion algorithms. In 2018, he received the Alexander Graham Bell Canada Graduate Scholarship-Master's (CGS-M) fund for his research work.

IVAN V. BAJIĆ (Senior Member, IEEE) is currently a Professor of engineering science and the Co-Director of the Multimedia Lab, Simon Fraser University, Burnaby, BC, Canada. His research interests include signal processing and machine learning with applications to multimedia processing, compression, and collaborative intelligence. He has authored about a dozen and coauthored another ten dozen publications in these fields. Several of his articles have received awards, most recently at ICIP 2019. He is an Elected Member of the IEEE Multimedia Signal Processing Technical Committee and the IEEE Multimedia Systems and Applications Technical Committee. He has served on the organizing and/or program committees for the main conferences in the field, and has received five reviewer awards, most recently at ICASSP 2019. He was an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA and the *IEEE Signal Processing Magazine*. He is also an Area Editor of *Signal Processing: Image Communication*.