Towards Extremely Compact RNNs for Video Recognition with Fully Decomposed Hierarchical Tucker Structure

Miao Yin[1], Siyu Liao[2†], Xiao-Yang Liu[3], Xiaodong Wang[3] and Bo Yuan[1]
[1]Rutgers University, [2]Amazon, [3]Columbia University
miao.yin@rutgers.edu, liasiyu@amazon.com, {xl2427, xw2008}@columbia.edu,
bo.yuan@soe.rutgers.edu

Abstract

Recurrent Neural Networks (RNNs) have been widely used in sequence analysis and modeling. However, when processing high-dimensional data, RNNs typically require very large model sizes, thereby bringing a series of deployment challenges. Although various prior works have been proposed to reduce the RNN model sizes, executing RNN models in the resource-restricted environments is still a very challenging problem. In this paper, we propose to develop extremely compact RNN models with fully decomposed hierarchical Tucker (FDHT) structure. The HT decomposition does not only provide much higher storage cost reduction than the other tensor decomposition approaches, but also brings better accuracy performance improvement for the compact RNN models. Meanwhile, unlike the existing tensor decomposition-based methods that can only decompose the input-to-hidden layer of RNNs, our proposed fully decomposition approach enables the comprehensive compression for the entire RNN models with maintaining very high accuracy. Our experimental results on several popular video recognition datasets show that, our proposed fully decomposed hierarchical tucker-based LSTM (FDHT-LSTM) is extremely compact and highly efficient. To the best of our knowledge, FDHT-LSTM, for the first time, consistently achieves very high accuracy with only few thousand parameters (3,132 to 8,808) on different datasets. Compared with the state-of-the-art compressed RNN models, such as TT-LSTM, TR-LSTM and BT-LSTM, our FDHT-LSTM simultaneously enjoys both order-of-magnitude (3,985× to 10,711×) fewer parameters and significant accuracy improvement (0.6% to 12.7%).

1. Introduction

Recurrent Neural Networks (RNNs), especially their advanced variants such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), have achieved unprecedented success in sequence analysis and processing. Thanks to their powerful capability of capturing and modeling the temporary dependency and correlation in the sequential data, the state-of-the-art RNNs have been widely deployed in many important artificial intelligence (AI) fields, such as natural language processing (NLP) [19], speech recognition [15], and computer vision [25].

Despite their current prosperity, the efficient deployment of RNNs is still facing several challenges, especially the *large model size problem*. Due to the widespread existence of high-dimensional input data in many applications, e.g. NLP and video processing, the input-to-hidden weight matrices of RNNs are often extremely large. For instance, as pointed out in [23], even with small-size hidden layer such as 256 hidden states, an LSTM working on UCF11 video recognition dataset [14] already requires more than 50 million parameters. Such ultra-high model size, consequently, brings a series of deployment challenges for RNNs, including but not limited to high difficulty of training, susceptibility to overfitting, long processing latency and inefficient energy consumption etc.

In order to address this problem, several prior works [4], [27] use convolutional neural network (CNNs) as the front-end feature extractors to reduce the size of input data of back-end RNNs. Although this strategy indeed reduces the model size as compared to the pure RNN-based solution, the back-end RNN part is still very large. Leveraging the popular model compression approaches in CNNs, such as pruning and quantization, is another alternative. However, the compression ratio provided by these methods is still insufficient, considering the very large size of the original RNN models.

Tensor Decomposition-based RNNs. Due to the above described limitations, recent RNN compression studies have mainly focused on a new direction – building the compact RNN models using low-rank *tensor decomposition*. By its nature, tensor decomposition can represent a very large tensor with the combination of multiple very small tensor

[†]This work was done when the author was with Rutgers University.

cores. Correspondingly, the number of the required representation parameters can be significantly reduced. Based on this unique and powerful capability, various compact RNN models have been developed using different tensor decomposition approaches. In [23], TT-LSTM and TT-GRU, which decompose input-to-hidden layer of RNNs to Tensor Train (TT) format, are proposed for video recognition. Similarly, BT-LSTM [24] and TR-LSTM [24] are also developed by decomposing the input-to-hidden layer via Block-Term Tensor (BT) and Tensor Ring (TR) formats, respectively. Besides those compact RNN models for computer vision tasks, in [26] a TT-structure LSTM is developed for long-term forecasting in dynamic systems. Compared with the original large-size RNNs, these compact models show significant model size reduction with maintaining competitive classification/prediction accuracy.

Limitations of Prior Works. Despite their promising potentials, the state-of-the-art tensor decomposition-based RNN models are still facing two inherent limitations: (1) Only the input-to-hidden layers, instead of the entire RNNs, are decomposed to small tensor cores. Consequently, there are still a large amount of parameters in the uncompressed hidden-to-hidden layers of RNNs, thereby making the large model size problem may still exist, especially in the resource-constrained scenarios (see Figure 2). Performing additional tensor decomposition on hidden-to-hidden layers is an alternative solution; however, as will be shown in the next section, straightforward decomposing on both input-to-hidden and hidden-to-hidden layers causes significant accuracy degradation. (2) The underlying tensor decomposition approaches used in the state-of-the-art compact RNN models have inherent constraints and limitations. For instance, TT decomposition requires the border tensor cores have to be rank-1, thereby directly hindering the representation power of TT-LSTM. Also, BT-LSTM models suffers computation overhead due to the extra flatten and permutation operations incurred by BT structure. More generally, from the perspective of tensor theory, neither TT, TR nor BT decomposition provides the best space complexity reduction. Consequently, the existing tensor decomposition-based solutions are still not ideal for designing high-accuracy ultra-compact RNN models.

Technical Preview & Benefits. To overcome these limitations, in this paper we propose to develop extremely compact RNN models with *fully decomposed hierarchical Tucker (FDHT) structure*. As shown in Figure 1, our proposed FDHT-structure RNN models have two main features. First, *Hierarchical Tucker* (HT) decomposition [7], a little explored but powerful tool for capturing and modeling the correlation and structure in high-dimensional data, is used to build the underlying RNN structure. Second, the entire RNN models, instead of one or few component layers, are constructed in the HT structure in a homogeneous

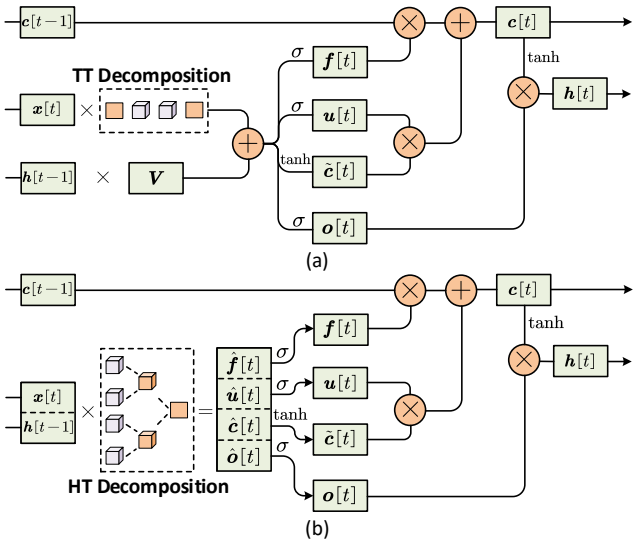

Figure 1: Architecture of tensor decomposition-based LSTM. $\boldsymbol{f}$ is the forget gate's activation vector, $\boldsymbol{u}$ is the input gate's activation vector, $\tilde{\boldsymbol{c}}$ is the cell input activation vector, and $\boldsymbol{o}$ is the output gate's activation vector in LSTM. (a) State-of-the-art TT-LSTM. (b) The proposed fully decomposed hiearchical Tucker LSTM (FDHT-LSTM).

way. Such exploration on the low-rank correlation among multiple component layers of RNN models is non-trivial, and brings order-of-magnitude parameter reduction over the state-of-the-art approaches with still maintaining very high accuracy. The benefits of the proposed FDHT-based RNN models are summarized as follows:

- **Benefits of Hierarchical Tucker Structure.** The underlying HT structure enables the RNN models enjoy much fewer model parameters and simultaneously higher accuracy than the state of the art. Compared with its well-explored counterparts (e.g. TT, TR and BT decomposition) adopted in the prior works, HT decomposition inherently provides higher space complexity reduction on the same-size tensor data with the same selected rank. By leveraging such theoretical advantage, the large-size RNN models can be constructed with very few parameters in the HT format. Additionally, the inherent hierarchical structure of HT also enables better weight sharing and hierarchical representation from high-dimensional data, thereby significantly improving RNN models' representation capability. As verified by our empirical experiments on different datasets, the LSTM models built on HT structure consistently outperform the existing LSTM models using other tensor decomposition in terms of both classification accuracy and model size reduction.

- **Benefits of Full Decomposition.** Built on the top of the underlying HT structure, enabling full decom-

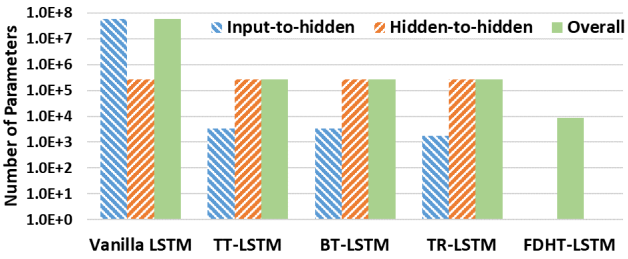

Figure 2: With compressing only input-to-hidden layer, hidden-to-hidden layer becomes the bottleneck of the state-of-the-art compact RNN models. The details of vanilla and compressed LSTM models are described in Table 2.

position with maintaining high accuracy brings further performance improvement for our proposed compact RNN models. As analyzed before, the existing tensor decomposition-based RNN models are limited by their uncompressed hidden-to-hidden layers. Although the same tensor decomposition can be performed on each individual uncompressed layers to further reduce model size, as will be shown in the experiment, such layer-wise decomposition causes significant accuracy degradation. Unlike this straightforward strategy, our proposed full decomposition approach integrates different RNN layers together, and then compresses the entire RNN model in a homogeneous way. Such integration-based full decomposition maintains the homogeneity of the model and avoid the accuracy loss. Consequently, our fully decomposed HT-structure RNN models further require even much fewer parameters than the non-fully decomposed HT-structure model with still remaining high accuracy.

Realizing Few Thousand Parameters-only RNN Models. By jointly using these two approaches, we develop extremely compact RNN models with high accuracy. Experiments show that, our proposed FDHT-LSTM consistently achieves very high accuracy with only very few parameters (3,132 to 8,808) on different video recognition datasets. To the best of our knowledge, it is the first RNN model that can only use few thousand parameters to achieve high accuracy on video recognition tasks. Compared with the state-of-the-art compressed RNN models, such as TT-LSTM, TR-LSTM and BT-LSTM, our FDHT-LSTM simultaneously enjoys both order-of-magnitude (3,985× to 10,711×) fewer parameters and significant accuracy improvement (0.6% to 12.7%).

Difference from Other Tucker/HT RNN/CNN Works. Recently in learning theory community, HT decomposition has been used to analyze the expressive power of RNNs and CNNs [16], [3] and [8]. However, these prior works focus on representing an **uncompressed** neural network models with HT format to explore the theoretical property such as expressive power. Instead, our work focus on building **compressed** RNN models via performing HT decomposition to its layers. Also, using Tucker decomposition to compress CNN/RNN is studied in [21], [10]. However, these works utilized Tucker decomposition, instead of Hierarchical Tucker decomposition, to compress models. In tensor theory, HT decomposition is different from Tucker decomposition, and HT enjoys much higher space complexity reduction than Tucker decomposition. Therefore, the compression ratio of our FDHT-RNN is much higher than the prior Tucker decomposition-based models.

2. Compact Fully Decomposed Hierarchical Tucker RNN Model

2.1. Preliminaries

Notation. Throughout the paper we use boldface calligraphic script letters, boldface capital letters, and boldface lower-case letters to represent tensors, matrices, and vectors, respectively, e.g. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}$, and $\boldsymbol{x} \in \mathbb{R}^{n_1}$. Also, $\boldsymbol{\mathcal{X}}_{(i_1,\cdots,i_d)} \in \mathbb{R}$ denotes the entry of tensor $\boldsymbol{\mathcal{X}}$. Similarly, $\boldsymbol{X}_{(i,j)}$ represents the entry of matrix $\boldsymbol{X}$.

Tensor Contraction. An HT-decomposed tensor is essentially the consecutive product of multiple tensor contraction results, where tensor contraction is executed between two tensors with at least one matched dimension. For instance, given two tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times l}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{l \times m_1 \times m_2}$, where the 3rd dimension of $\boldsymbol{\mathcal{A}}$ matches the 1st dimension of $\boldsymbol{\mathcal{B}}$ with length l, the tensor contraction result is a size- $n_1 \times n_2 \times m_1 \times m_2$ tensor as

$$(\boldsymbol{\mathcal{A}} \times_1^3 \boldsymbol{\mathcal{B}})_{(i_1,i_2,j_1,j_2)} = \sum_{\alpha=1}^{l} \boldsymbol{\mathcal{A}}_{(i_1,i_2,\alpha)} \boldsymbol{\mathcal{B}}_{(\alpha,j_1,j_2)}.$$

Hierarchical Tucker Decomposition. The Hierarchical Tucker decomposition is a special type of tensor decomposition approach with hierarchical levels with respect to the order of the tensor. As illustrated in Figure 3, an HT-decomposed tensor can be recursively decomposed into intermediate components, referred as *frames*, from top to bottom in a binary tree, where each frame corresponds to a unique *node*, and each node is associated with a *dimension set*. In general, for a HT-decomposed tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, we can build a binary tree with a root node associated with $D = \{1, 2, \cdots, d\}$ and $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}_D$ as the root frame. We define $s \subsetneq D$ is associated with the node corresponding to $\boldsymbol{\mathcal{U}}_s$, and $s_1, s_2 \subsetneq s$ are associated with the left and right child nodes of the s-associated node. Hence, as $\mu_s = \min(s), \nu_s = \max(s)$, each non-leaf frame $\boldsymbol{\mathcal{U}}_s \in \mathbb{R}^{r_s \times n_{\mu_s} \times \cdots \times n_{\nu_s}}$ can be recursively decomposed to its left and right child frames ($\boldsymbol{\mathcal{U}}_{s_1}$ and $\boldsymbol{\mathcal{U}}_{s_2}$) and transfer tensor $\boldsymbol{\mathcal{G}}_s \in \mathbb{R}^{r_s \times r_{s_1} \times r_{s_2}}$ as

$$\boldsymbol{\mathcal{U}}_s = \boldsymbol{\mathcal{G}}_s \times_1^2 \boldsymbol{\mathcal{U}}_{s_1} \times_1^2 \boldsymbol{\mathcal{U}}_{s_2}. \quad (1)$$

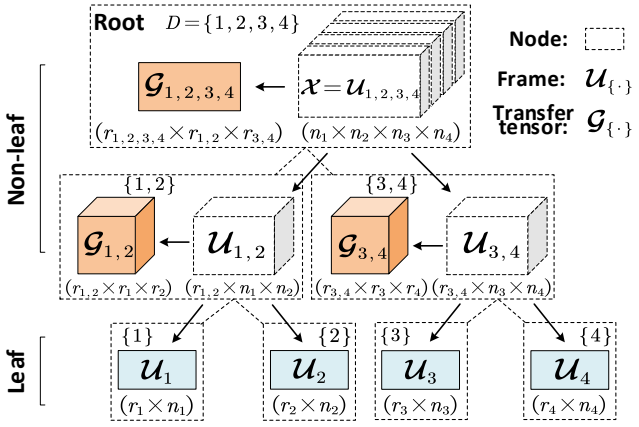

Figure 3: Example HT decomposition with a tensor of 4 orders. All the dashed lines and boxes describe a binary tree with root $D = \{1, 2, 3, 4\}$, where the dashed boxes represent the nodes. Here node $\{1\}$ is a leaf node, whose parent and sibling are node $\{1, 2\}$ and node $\{2\}$, respectively. Here $\boldsymbol{\mathcal{X}}$ is decomposed to a set of orange-colored transfer tensors and blue-colored leaf frames.

Consequently, by performing this recursive decomposition till the bottom of the binary tree, we can decompose the original $n_1 \times \cdots \times n_d$-order tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}_D$ into the combination of the 2-order leaf frames and 3-order transfer tensors. Notice that here r_s, as *hierarchical rank*, is an important parameter that determines the decomposition effect.

2.2. Compact HT-structure Linear Layer

In this subsection we describe the details of building a compact HT-structure linear layer, which will be used as the foundation to build the compact FDHT-RNN models.

Tensorization. In general, the key idea of building HT structured linear layer is to transform the weight matrix $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ to the HT-based format. Considering $\boldsymbol{W}$ is a 2-D matrix, while HT decomposition is mainly performed on high-order tensor, we first need to reshape $\boldsymbol{W}$ as well as its affiliated input vector $\boldsymbol{x} \in \mathbb{R}^N$ and output vector $\boldsymbol{y} \in \mathbb{R}^M$ to tensor format as $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1\times\cdots\times m_d\times n_1\times\cdots\times n_d}$, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{m_1\times\cdots\times m_d}$, respectively, where $M = \prod_{i=1}^{d} m_i$ and $N = \prod_{j=1}^{d} n_j$.

Decomposing $\boldsymbol{W}$. Given a tensorized $\boldsymbol{W}$ as $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1\times\cdots\times m_d\times n_1\times\cdots\times n_d}$, we can now leverage HT decomposition to represent the large-size $\boldsymbol{W}$ using a set of small-size matrices and tensors. In general, following Equation (1), $\boldsymbol{\mathcal{W}}$ can be decomposed as

$$\boldsymbol{\mathcal{W}}_{(i_1,\cdots,i_d,j_1,\cdots,j_d)} = \sum_{k=1}^{r_D}\sum_{p=1}^{r_{D_1}}\sum_{q=1}^{r_{D_2}}(\boldsymbol{\mathcal{G}}_D)_{(k,p,q)} \cdot(\boldsymbol{\mathcal{U}}_{D_1})_{(p,\varphi_{D_1}(\boldsymbol{i},\boldsymbol{j}))}(\boldsymbol{\mathcal{U}}_{D_2})_{(q,\varphi_{D_2}(\boldsymbol{i},\boldsymbol{j}))}, \tag{2}$$

where $\varphi_s(\boldsymbol{i}, \boldsymbol{j})$ is a mapping function that produces the correct indices $\boldsymbol{i} = (i_1, \cdots, i_d)$ and $\boldsymbol{j} = (j_1, \cdots, j_d)$ for a specified frame $\boldsymbol{\mathcal{U}}_s$ with the given s and d. For instance, with $d = 6$ and $s = \{3, 4\}$, the output of $\varphi_s(\boldsymbol{i}, \boldsymbol{j})$ is (i_3, i_4, j_3, j_4). In addition, $\boldsymbol{\mathcal{U}}_{D_1}$ and $\boldsymbol{\mathcal{U}}_{D_2}$ can be recursively computed as

$$(\boldsymbol{\mathcal{U}}_s)_{(k,\varphi_s(\boldsymbol{i},\boldsymbol{j}))} = \sum_{p=1}^{r_{s_1}}\sum_{q=1}^{r_{s_2}}(\boldsymbol{\mathcal{G}}_s)_{(k,p,q)} \cdot(\boldsymbol{\mathcal{U}}_{s_1})_{(p,\varphi_{s_1}(\boldsymbol{i},\boldsymbol{j}))}(\boldsymbol{\mathcal{U}}_{s_2})_{(q,\varphi_{s_1}(\boldsymbol{i},\boldsymbol{j}))}, \tag{3}$$

where $D = \{1, 2, \cdots, d\}$, $D_1 = \{1, \cdots, \lfloor d/2 \rfloor\}$ and $D_2 = \{\lfloor d/2 \rfloor, \cdots, d\}$ are associated with left and right child nodes of the root node.

HT-structure Layer. With the HT-decomposed weight matrix, a linear layer with HT structure can now be developed. Specifically, the HT-format matrix-vector multiplication, as the kernel computation in the forward propagation procedure on the layer, is performed as follows:

$$\boldsymbol{\mathcal{Y}}_{(\boldsymbol{i})} = \sum_{\boldsymbol{j}}\sum_{k=1}^{r_D}\sum_{p=1}^{r_{D_1}}\sum_{q=1}^{r_{D_2}}(\boldsymbol{\mathcal{G}}_D)_{(k,p,q)} \cdot(\boldsymbol{\mathcal{U}}_{D_1})_{(p,\varphi_{D_1}(\boldsymbol{i},\boldsymbol{j}))}(\boldsymbol{\mathcal{U}}_{D_2})_{(q,\varphi_{D_2}(\boldsymbol{i},\boldsymbol{j}))}\boldsymbol{\mathcal{X}}_{(\boldsymbol{j})}. \tag{4}$$

Considering the desired output $\boldsymbol{y}$ of HT-structure layer is a vector, the calculated $\boldsymbol{\mathcal{Y}}$ needs to be re-shaped again to the 1-D format. Consequently, we denote the entire forward computing procedure from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ as

$$\boldsymbol{y} = HTL(\boldsymbol{W}, \boldsymbol{x}). \tag{5}$$

Figure 4 illustrates the computation process in an HT-structure layer. Here the arrows in a sequence represent the whole computation flow. As shown in this figure, the input vector is first tensorized, and then tensor contraction is performed in a hierarchical way. Finally the tensorized output is obtained.

Benefits on Low Cost. One major benefit of using HT-structure linear layer is that its inherent high complexity reduction. As shown in Table 1, compared with the vanilla uncompressed linear layer as well as other tensor decomposition-based layers, using HT structure can bring the lower space complexity with the same rank setting. Besides theoretical complexity analysis, we also verify this low-cost benefits of HT structure via empirical experiments. Figure 5 shows the number of parameters to store a compact weight matrix. Here we adopt the size-$57,600 \times 256$ weight matrix used in [23] [24] [17] for evaluation. From Figure 5 it is seen that HT-based approach indeed require the fewest parameters than other tensor decomposition-based methods. Our evaluation results in the Experiments Section also verify such advantages on compression ratios over various datasets.

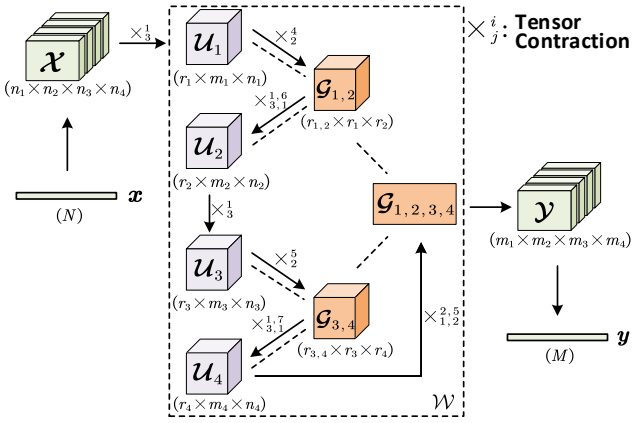

Figure 4: Example computation process in an HT-structure layer with d=4. Solid lines with arrow connect two tensors that are contracted. Note that different from Figure 3, the leaf frames $\mathcal{U}$ here is 3-dimensional. This is because the dimensions of input tensor $\mathcal{X}$ here is different.

Compressed Linear Layer	Space Complexity
Uncompressed	$\mathcal{O}(NM)$
Tensor Train (TT)-structure	$\mathcal{O}(dmnr^2)$
Tensor Ring (TR)-structure	$\mathcal{O}(dmnr^2)$
Block-Term (BT)-structure	$\mathcal{O}(dmnr + r^d)$
Hierarchical Tucker (HT)-structure	$\mathcal{O}(dmnr + dr^3)$

Table 1: Comparison of space complexity among different tensor decomposition-based linear layers. $r = \max_{s \subsetneq D} r_s$, $m = \max_{k \in D} m_k$, $n = \max_{k \in D} n_k$.

2.3. Fully Decomposing the HT-structure RNN

Challenges of Fully Compressing RNN. Based on the proposed HT-structure layer, next we aim to compress the entire RNN models using HT decomposition. A straightforward way is to simply build each component layer of RNNs with HT format. In other words, all of the weight matrices of input-to-hidden and hidden-to-hidden layers are decomposed into HT structure. Although indeed providing further compression on the hidden-to-hidden layers, such layer-wise compression strategy suffers huge accuracy drop. As shown in Figure 6, the layer-wise compression using HT decomposition causes 2.4% accuracy drop as compared to the input-to-hidden-only compression. Therefore, realizing the full compression of the entire RNN models without accuracy drop is non-trivial but challenging.

Proposed FDHT-RNN. To achieve that, we propose to develop the entire compact RNNs in a homogeneous way, namely *fully decomposed HT (FDHT)* structure. Take LSTM, as the most popular and advanced variant of RNNs, for example. As illustrated in Figure 1, our key idea is to first concatenate all the weight matrix as a single big matrix. In other words, the entire LSTM model can be interpreted as a single "mega" linear layer. From the perspective of forward propagation, at each time step, all the intermediate results can be viewed as being calculated using only one matrix multiplication:

$$\begin{bmatrix} \hat{\boldsymbol{f}}[t] \\ \hat{\boldsymbol{u}}[t] \\ \hat{\boldsymbol{c}}[t] \\ \hat{\boldsymbol{o}}[t] \end{bmatrix} = \begin{bmatrix} \boldsymbol{W}_f & \boldsymbol{V}_f \\ \boldsymbol{W}_u & \boldsymbol{V}_u \\ \boldsymbol{W}_c & \boldsymbol{V}_c \\ \boldsymbol{W}_o & \boldsymbol{V}_o \end{bmatrix} \begin{bmatrix} \boldsymbol{x}[t] \\ \boldsymbol{h}[t-1] \end{bmatrix} = \boldsymbol{W}\boldsymbol{I}[t]. \tag{6}$$

Based on the above interpretation, the HT structure can be imposed on this integrated layer. Specifically, we can tensorize and decompose the entire RNN models to HT format and perform forward propagation as follows:

$$\begin{bmatrix} \hat{\boldsymbol{f}}[t] \\ \hat{\boldsymbol{u}}[t] \\ \hat{\boldsymbol{c}}[t] \\ \hat{\boldsymbol{o}}[t] \end{bmatrix} = \boldsymbol{Z}[t] = HTL(\boldsymbol{W}, \boldsymbol{I}[t]). \tag{7}$$

After this HT-based computation, the outputs of the FDHT-

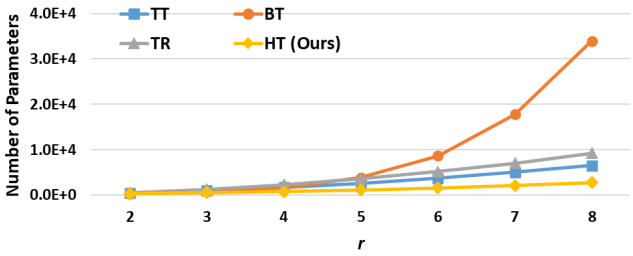

Figure 5: Comparison on number of parameters with different tensor decomposition for the same weight matrix. All the tensor decomposition methods use the same setting $d = 5$, $(n_1, \cdots, n_5) = (8, 10, 10, 9, 8)$, $(m_1, \cdots, m_5) = (4, 4, 2, 4, 2)$, and r is the rank.

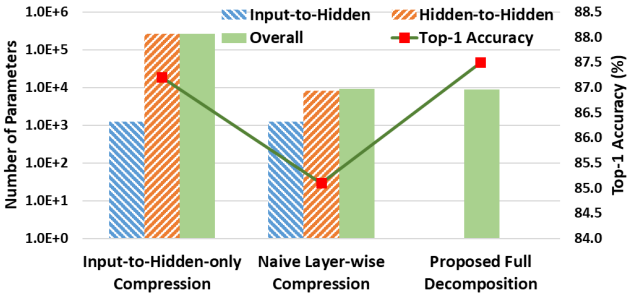

Figure 6: Comparison of number of parameters and accuracy between layer-wise compression and the proposed full decomposition. Evaluation dataset is UCF11.

LSTM can be calculated as follows:

$$\begin{aligned}\boldsymbol{f}[t] =&\sigma(\hat{\boldsymbol{f}}[t] + \boldsymbol{b}_f)\\ \boldsymbol{u}[t] =&\sigma(\hat{\boldsymbol{u}}[t] + \boldsymbol{b}_u)\\ \boldsymbol{c}[t] =&\boldsymbol{f}[t] \odot \boldsymbol{c}[t-1] + \boldsymbol{u}[t] \odot \tanh(\hat{\boldsymbol{c}}[t] + \boldsymbol{b}_c)\\ \boldsymbol{o}[t] =&\sigma(\hat{\boldsymbol{o}}[t] + \boldsymbol{b}_o)\\ \boldsymbol{h}[t] =&\boldsymbol{o}[t] \odot \tanh(\boldsymbol{c}[t]),\end{aligned} \tag{8}$$

where σ, $\tanh$ and $\odot$ are the sigmoid function, hyperbolic function and element-wise product, respectively.

HT-based Gradient Calculation. To ensure the valid training on FDHT-RNN, the gradient calculation in the backward propagation should also be accordingly reformulated to HT-based format. In general, considering for HT-structure linear layer $\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{U}}_D$ and $\frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_D} = \boldsymbol{\mathcal{X}}$, assuming s is associated with a left node, as we denote that $F(s)$ and $B(s)$ are the sets associated with the parent and sibling nodes of the s-associated node in the binary tree, respectively, and define $\mu_s = \min(s), \nu_s = \max(s)$, the partial derivative of output tensor with respect to frames can be calculated in the following recursive way until $F(s)$ is equal to D:

$$\begin{aligned}\frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} =&\boldsymbol{\mathcal{G}}_{F(s)} \times_1^3 \boldsymbol{\mathcal{U}}_{B(s)}\\ &\times_{\substack{1,3,\cdots,2\nu_{B(s)}-2\mu_{B(s)}+4\\ 1,\cdots,\nu_{B(s)}-\mu_{B(s)}+2,\nu_{F(s)}-\mu_{F(s)}+3,\\ \cdots,\nu_{F(s)}-\mu_{F(s)}+\nu_{B(s)}-\mu_{B(s)}+3}} \frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_{F(s)}}.\end{aligned} \tag{9}$$

Based on Equation (9), the gradients for leaf frames and transfer tensors can be computed as follows:

$$\frac{\partial L}{\partial \boldsymbol{\mathcal{U}}_s} = \frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} \times_{1,\cdots,\mu_s-1,\nu_s+1,\cdots,d}^{\nu_s-\mu_s+3,\cdots,d+1} \frac{\partial L}{\partial \boldsymbol{\mathcal{Y}}}, \tag{10}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\mathcal{G}}_s} =&\frac{\boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} \times_{2,\cdots,\nu_{s_1}-\mu_{s_1}+2}^{2,\cdots,\nu_{s_1}-\mu_{s_1}+2} \boldsymbol{\mathcal{U}}_{s_1}\\ &\times_{2,\cdots,\nu_{s_2}-\mu_{s_2}+2}^{3,\cdots,\nu_{s_2}-\mu_{s_2}+3} \boldsymbol{\mathcal{U}}_{s_2} \times_{1,\cdots,d}^{4,\cdots,d+3} \frac{\partial L}{\partial \boldsymbol{\mathcal{Y}}}.\end{aligned} \tag{11}$$

In general, our proposed "integrate-then-decompose" strategy brings significant performance benefit for the compact FDHT-RNN models. As illustrated in Figure 6, unlike layer-wise compression that has significant accuracy drop, our proposed full decomposition approach does not bring any performance loss. Instead, it even outperforms the input-to-hidden-only compression counterpart with much fewer parameters. As will be shown in the next section, evaluation on various datasets also show that our full decomposed HT-format RNN models achieve the same or even higher accuracy than the model that has only HT structure on the input-to-hidden layers. We hypothesize such phenomenon may result from the special architecture of FDHT-RNN: when the entire RNN is constructed in the HT structure, the multi-dimensional low-rank correlations across the entire model can be explored and captured in a more precise and comprehensive way, thereby enabling very compact RNN model without affecting accuracy performance.

3. Experiments

In this section, we evaluate the performance of FDHT-RNN on different datasets, and compare them with the state-of-the-art with respect to compression ratio and test accuracy. Considering LSTM is the current most commonly used RNN variant in both academia and industry, our experiments focus on FDHT-LSTM, and compare it with the vanilla uncompressed LSTM and recent advances in compressed RNN such as TT-LSTM [23], BT-LSTM [24] and TR-LSTM [17]. Also, similar to prior works, we also train and evaluate a model that only has HT structure in the input-to-hidden layer, namely HT-LSTM, for ablation study.

Selection of Tensor Parameters d and r. In our experiments we set d, as the dimension number of the tensors that the input, output and weight matrix are tensorized to, as the same to the setting in [23] and [24]. Also similar to prior works, we select r via empirical setting to make good balance between compression ratio and model accuracy.

Training Strategy. Following the similar setting in prior works, we adopt two types of training strategy: end-to-end direct training and training with pre-trained CNN. In the end-to-end direct training the input of LSTM is the raw data, e.g. video clips; while training with pre-trained CNNs means the back-end LSTM receives the compact features extracted by a front-end pre-trained CNN.

3.1. End-to-End Direct Training (RNN-only Model)

Hyperparameter Setting. We train the models using ADAM optimizer with L2 regularization of coefficient 0.001. Also, dropout rate is set as 0.25 and batch size is 16.

UCF11 Dataset. The UCF11 dataset [14] consists of 11-class human action (e.g. biking, diving, basketball) videos with totally 1,600 video clips. Each class is assembled by 25 video groups, where each group contains at least 4 action clips with resolution as 320×240.

At data pre-processing stage we choose the same settings used in the related work [24] [17] for fair comparison. Specifically, the resolution of video clips is first scaled down to 160×120, and then 6 frames from each clip are randomly sampled to form the sequential input.

For the baseline vanilla uncompressed LSTM model, it contains 4 input-to-hidden layers and 4 hidden-to-hidden layers, where the size of its input vector is $160 \times 120 \times 3 = 57,600$, and the number of hidden states in each layer is 256. For our proposed FDHT-LSTM, in order to factorize the size of concatenated vector $\boldsymbol{I}$, we pad the input vector to size $16 \times 16 \times 16 \times 15 - 256 = 61184$ by zeros. The concatenated vector is reshaped to a tensor of shape $16 \times 16 \times 16 \times 15$, and the hidden state is reshaped to a

Model	Number of Parameters		Top-1 Acc. (%)
	Input-Hidden	Overall	
LSTM	58.98M	59.24M	69.7
TT-LSTM	3,360	265.50K (223×)	79.6
BT-LSTM	3,387	265.53K (223×)	85.3
TR-LSTM	1,725	263.87K (225×)	86.9
HT-LSTM (Ours)	1,245	263.39K (225×)	87.2
FDHT-LSTM (Ours)	-	**8,808 (6,726×)**	**87.5**

Table 2: Performance of different RNN compression works on UCF11 dataset using end-to-end direct training.

tensor of shape $4 \times 4 \times 4 \times 4$. All leaf and non-leaf ranks are set as 14 and 12, respectively.

Table 2 summarizes the performance of our FDHT-LSTM on UCF11 dataset and compare it with the related works. It is seen that compared with vanilla LSTM using 59 million parameters, FDHT-LSTM only needs 8,808 parameters with 17.8% accuracy increase. Compared with the recent advances on compressing RNNs using other tensor decomposition methods, including TT-LSTM, BT-LSTM and TR-LSTM, our proposed FDHT-LSTM requires at least 6,501× fewer parameters with at least 0.6% accuracy increase.

Youtube Celebrities Face Dataset. Youtube dataset [9] contains 1,910 video clips from 47 subjects. Also, the resolutions of the frames vary for different video clips. Being consistent with prior works, for data pre-processing the resolution of the input data to FDHT-LSTM is re-scaled as 160×120. Also, 6 frames in each video clips are randomly sampled to form the input sequence.

In this experiment we build a FDHT-LSTM with the similar setting with the one used in UCF11 dataset. A slight difference is here all non-leaf ranks are set as 11. Table 3 shows the performance comparison with TT-LSTM [23] on this dataset. From this table it is seen that FDHT-LSTM achieves 7,117× compression ratio over the original uncompressed LSTM with much higher accuracy. Compared with TT-LSTM, FDHT-LSTM has 6,894× fewer parameters with 12.7% higher accuracy.

Besides, on the same Youtube dataset we also compare FDHT-LSTM with several other reported works without using tensor decomposition method. As shown in Table 4, among those works the state-of-the-art model is [12], which has the highest reported accuracy (84.6%). Compared with that model, FDHT-LSTM achieves 3.6% higher test accuracy with using much fewer parameters.

Model	Number of Parameters		Top-1 Acc. (%)
	Input-Hidden	Overall	
LSTM	58.98M	59.24M	33.2
TT-LSTM	3,392	265.54K (223×)	75.5
HT-LSTM (Ours)	810	262.95K (225×)	88.1
FDHT-LSTM (Ours)	-	**8,324 (7,117×)**	**88.2**

Table 3: Performance of different RNN compression work on Youtube celebrities face dataset using end-to-end direct training. **Notice that no prior works report performance of BT-LSTM and TR-LSTM on this dataset.**

Model	Number of Parameters	Top-1 Acc. (%)
DML-PV	220K	82.8
VGGFACE + RRNN	⩾42M	84.6
VGG16-GCR	138M	82.9
HT-LSTM (Ours)	263K	88.1
FDHT-LSTM (Ours)	**8,324**	**88.2**

Table 4: Comparison between HT-LSTM using end-to-end direct training and other models without using tensor decomposition on Youtube celebrities face dataset, such as DML-PV[2], VGGFACE + RRNN [12] and VGG16-GCR [13].

3.2. Training with Pre-trained CNNs (CNN+RNN)

Another set of our experiments is based on training strategy using pre-trained CNNs as the front-end feature extractor. As indicated in [4], using the front-end CNN can reduce the required input vector size of RNN and significantly improve the overall performance of the entire CNN+RNN model.

Hyperparameter Setting. In this part of experiments dropout rate is set as 0.5. The L2 regularization with coefficient 0.0001 is used, and the entire FDHT-LSTM model is trained using ADAM optimizer with batch size 16.

UCF11 Dataset. Consistent with [17], Inception-V3 [20] is selected as the the front-end CNN, whose output is a flattened size-2,048 feature vector. For the baseline vanilla uncompressed LSTM model, the size of hidden state is set as 2048. For our proposed FDHT-LSTM, the concatenated vector is of size $2048 + 2048 = 4096$, and is reshaped to a tensor of size $8 \times 8 \times 8 \times 8$. Similarly, the output vector is reshaped to a tensor of size $4 \times 8 \times 8 \times 8$. In addition, all leaf ranks are set as 9, and all non-leaf ranks are set as 6.

Model	Number of Parameters	Top-1 Acc. (%)
Soft Attention	311M	85.0
Deep Fusion	179M	94.6
CNN + LSTM	55M	92.3
CNN + TR-LSTM	39M	93.8
CNN + HT-LSTM (Ours)	22M	98.1
CNN + FDHT-LSTM (Ours)	**22M**	**98.4**

Table 5: Comparison between FDHT-LSTM using front-end pre-trained CNN and other related works on UCF11 dataset, such as Soft Attention [18], Deep Fusion [6] and CNN + LSTM/TR-LSTM [17]. **Notice that no prior works report performance of CNN + TT-LSTM and CNN + BT-LSTM on this dataset.**

Table 5 summarizes the test accuracy of different models over UCF11 dataset. It is seen that with pre-trained CNN model as front-end feature extractor, FDHT-LSTM achieves 98.4% accuracy, which is 3.8% higher than the best reported accuracy from the state of the art. Compared with the TR-LSTM using the same front-end CNN, FDHT-LSTM achieves 4.6% accuracy increase. Meanwhile, as shown in Table 7, for such CNN + RNN model, TR-LSTM only brings 1.9× fewer parameters, while FDHT achieves 2.5× parameter reduction.

HMDB51 Dataset. HMDB51 dataset [11] contains 6,849 video clips that belong to 51 action categories, where each of them consists of more than 101 clips. Again, for the experiment on this dataset we use Inception-V3 as the pre-trained CNN model, whose extracted feature is flattened to a length-2,048 vector. Reshaped from concatenated vector, the tensor has size of $8 \times 8 \times 8 \times 8$. We also reshape the output vector to a tensor of size $4 \times 8 \times 8 \times 8$. Meanwhile, all leaf ranks are set as 14, and all non-leaf ranks are set as 12.

Table 6 summarizes the performance of CNN-aided FDHT-LSTM and other related works on this dataset. It is seen that FDHT-LSTM achieves 64.2% accuracy, which obtains 0.4% increase than the recent TR-LSTM. Note that two-stream I3D is a 3D-CNN model which is more advanced, while our model only uses 2D pre-trained CNN. With fewer parameters, our approach can still achieve the comparable performance. As shown in Table 7, for such CNN+RNN model, TR-LSTM only brings 1.4× fewer parameters, while FDHT-LSTM achieves 2.5× parameter reduction.

4. Conclusion

In this paper, we propose a new and extremely compact RNN model with fully decomposed and Hierarchical Tucker structure. Our experiments on different datasets show that, our proposed FDHT-LSTM models significantly outperform the state-of-the-art compressed RNN models in terms of both compression ratio and test accuracy. To the best of our knowledge, FDHT-LSTM is the first RNN model that achieves very high accuracy with only few thousand parameters on different video recognition datasets.

Model	Number of Parameters		
	TDD + FV	117M	63.2
VGG + Two-Stream Fusion	181M	62.1
Two-Stream I3D	25M	**66.4**
CNN + LSTM	55M	62.9
CNN + TR-LSTM	39M	63.8
CNN + HT-LSTM (Ours)	22M	64.2
CNN + FDHT-LSTM (Ours)	**22M**	64.2

Table 6: Comparison among HT-LSTM using front-end pre-trained CNN and other related works on HMDB51 dataset, such as TDD + FV [22], VGG + Two-Stream Fusion [5], Two-Stream I3D [1] and CNN + LSTM/TR-LSTM [17]. **Notice that no prior works report performance of CNN + TT-LSTM and CNN + BT-LSTM on this dataset.**

Model (with CNN)	Number of Parameters			
	CNN	RNN		Total
		Input-Hidden	Overall	
LSTM	21.77M	16.78M	33.55M	55.32M
TR-LSTM		671.2K	17.45M (1.9×)	39.22M (1.4×)
FDHT-LSTM (UCF11)		-	**3,132 (10,713×)**	**21.77M (2.5×)**
FDHT-LSTM (HMDB51)		-	**8,416 (3,987×)**	**21.78M (2.5×)**

Table 7: Comparison of detailed model size among our FDHT-LSTM, TR-LSTM and vanilla LSTM with pre-trained CNN. The same models of CNN + LSTM and CNN + TR-LSTM are used for both UCF11 and HMDB51 datasets.

Acknowledgements

This work was partially supported by National Science Foundation under Grant CCF-1854742 and SHF-7995357.

References

[1] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017. 8

[2] Gong Cheng, Peicheng Zhou, and Junwei Han. Duplex metric learning for image set classification. *IEEE Transactions on Image Processing*, 27(1):281–292, 2017. 7

[3] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016. 3

[4] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015. 1, 7

[5] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016. 8

[6] Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Two stream lstm: A deep fusion framework for human action recognition. In *IEEE Winter Conference on Applications of Computer Vision*, pages 177–186. IEEE, 2017. 8

[7] Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, 2009. 2

[8] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017. 3

[9] Minyoung Kim, Sanjiv Kumar, Vladimir Pavlovic, and Henry Rowley. Face tracking and recognition with visual constraints in real-world videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008. 7

[10] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 3

[11] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *International Conference on Computer Vision*, pages 2556–2563. IEEE, 2011. 8

[12] Yang Li, Wenming Zheng, Zhen Cui, and Tong Zhang. Face recognition based on recurrent regression neural network. *Neurocomputing*, 297:50–58, 2018. 7

[13] Bo Liu, Liping Jing, Jia Li, Jian Yu, Alex Gittens, and Michael W Mahoney. Group collaborative representation for image set classification. *International Journal of Computer Vision*, 127(2):181–206, 2019. 7

[14] Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos "in the wild". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1996–2003. IEEE, 2009. 1, 6

[15] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5528–5531. IEEE, 2011. 1

[16] Cohen Nadav, Sharir Or, Levine Yoav, Tamari Ronen, Yakira David, and Shashua Amnon. Analysis and design of convolutional networks via hierarchical tensor decompositions. *arXiv preprint arXiv:1705.02302*, 2018. 3

[17] Yu Pan, Jing Xu, Maolin Wang, Jinmian Ye, Fei Wang, Kun Bai, and Zenglin Xu. Compressing recurrent neural networks with tensor ring for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4683–4690, 2019. 4, 6, 7, 8

[18] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. In *Neural Information Processing Systems: Time Series Workshop*, 2015. 8

[19] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *International Conference on Machine Learning*, pages 1017–1024, 2011. 1

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. 7

[21] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Tensor decomposition for compressing recurrent neural network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018. 3

[22] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4305–4314, 2015. 8

[23] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*, pages 3891–3900. JMLR. org, 2017. 1, 2, 4, 6, 7

[24] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9378–9387, 2018. 2, 4, 6

[25] Haonan Yu, Jiang Wang, Zhiheng Huang, Yi Yang, and Wei Xu. Video paragraph captioning using hierarchical recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4584–4593, 2016. 1

[26] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using higher order tensor rnns. *arXiv preprint arXiv:1711.00073*, 2017. 2

[27] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition, pages 4694–4702, 2015. 1