Tensor-Train Recurrent Neural Networks for Video Classification

Yinchong Yang [1 2] **Denis Krompass** [2] **Volker Tresp** [1 2]

Abstract

The Recurrent Neural Networks and their variants have shown promising performances in sequence modeling tasks such as Natural Language Processing. These models, however, turn out to be impractical and difficult to train when exposed to very high-dimensional inputs due to the large input-to-hidden weight matrix. This may have prevented RNNs' large-scale application in tasks that involve very high input dimensions such as video modeling; current approaches reduce the input dimensions using various feature extractors. To address this challenge, we propose a new, more general and efficient approach by factorizing the input-to-hidden weight matrix using Tensor-Train decomposition which is trained simultaneously with the weights themselves. We test our model on classification tasks using multiple real-world video datasets and achieve competitive performances with state-of-the-art models, even though our model architecture is orders of magnitude less complex. We believe that the proposed approach provides a novel and fundamental building block for modeling high-dimensional sequential data with RNN architectures and opens up many possibilities to transfer the expressive and advanced architectures from other domains such as NLP to modeling high-dimensional sequential data.

1. Introduction

Nowadays, the Recurrent Neural Network (RNN), especially its more advanced variants such as the LSTM and the GRU, belong to the most successful machine learning approaches when it comes to sequence modeling. Especially in Natural Language Processing (NLP), great improvements have been achieved by exploiting these Neural Network architectures. This success motivates efforts to also apply these RNNs to video data, since a video clip could be seen as a sequence of image frames. However, plain RNN models turn out to be impractical and difficult to train directly on video data due to the fact that each image frame typically forms a relatively high-dimensional input, which makes the weight matrix mapping from the input to the hidden layer in RNNs extremely large. For instance, in case of an RGB video clip with a frame size of say $160\times120\times3$, the input vector for the RNN would already be $57,600$ at each time step. In this case, even a small hidden layer consisting of only 100 hidden nodes would lead to 5,760,000 free parameters, only considering the input-to-hidden mapping in the model.

In order to circumvent this problem, state-of-the-art approaches often involve pre-processing each frame using Convolution Neural Networks (CNN), a Neural Network model proven to be most successful in image modeling. The CNNs do not only reduce the input dimension, but can also generate more compact and informative representations that serve as input to the RNN. Intuitive and tempting as it is, training such a model from scratch in an end-to-end fashion turns out to be impractical for large video datasets. Thus, many current works following this concept focus on the CNN part and reduce the size of RNN in term of sequence length (Donahue et al., 2015; Srivastava et al., 2015), while other works exploit pre-trained deep CNNs as pre-processor to generate static features as input to RNNs (Yue-Hei Ng et al., 2015; Donahue et al., 2015; Sharma et al., 2015). The former approach neglects the capability of RNNs to handle sequences of variable lengths and therefore does not scale to larger, more realistic video data. The second approach might suffer from suboptimal weight parameters by not being trained end-to-end (Fernando & Gould, 2016). Furthermore, since these CNNs are pre-trained on existing image datasets, it remains unclear how well the CNNs can generalize to video frames that could be of totally different nature from the image training sets.

Alternative approaches were earlier applied to generate image representations using dimension reductions such as PCA (Zhang et al., 1997; Kambhatla & Leen, 1997; Ye et al., 2004) and Random Projection (Bingham & Mannila, 2001). Classifiers were built on such features to perform object and face recognition tasks. These models, however,

[1]Ludwig Maximilian University of Munich, Germany [2]Siemens AG, Corporate Technology, Germany. Correspondence to: Yinchong Yang <yinchong.yang@siemens.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

are often restricted to be linear and cannot be trained jointly with the classifier.

In this work, we pursue a new direction where the RNN is exposed to the raw pixels on each frame without any CNN being involved. At each time step, the RNN first maps the large pixel input to a latent vector in a typically much lower dimensional space. Recurrently, each latent vector is then enriched by its predecessor at the last time step with a hidden-to-hidden mapping. In this way, the RNN is expected to capture the inter-frame transition patterns to extract the representation for the entire sequence of frames, analogous to RNNs generating a sentence representation based on word embeddings in NLP (Sutskever et al., 2014). In comparison with other mapping techniques, a direct input-to-hidden mapping in an RNN has several advantages. First it is much simpler to train than deep CNNs in an end-to-end fashion. Secondly it is exposed to the complete pixel input without the linear limitation as PCA and Random Projection. Thirdly and most importantly, since the input-to-hidden and hidden-to-hidden mappings are trained jointly, the RNN is expected to capture the correlation between spatial and temporal patterns.

To address the issue of having too large of a weight matrix for the input-to-hidden mapping in RNN models, we propose to factorize the matrix with the Tensor-Train decomposition (Oseledets, 2011). In (Novikov et al., 2015) the Tensor-Train has been applied to factorize a fully-connected feed-forward layer that can consume image pixels as well as latent features. We conducted experiments on three large-scale video datasets that are popular benchmarks in the community, and give empirical proof that the proposed approach makes very simple RNN architectures competitive with the state-of-the-art models, even though they are of several orders of magnitude lower complexity.

The rest of the paper is organized as follows: In Section 2 we summarize the state-of-the-art works, especially in video classification using Neural Network models and the tensorization of weight matrices. In Section 3 we first introduce the Tensor-Train model and then provide a detailed derivation of our proposed Tensor-Train RNNs. In Section 4 we present our experimental results on three large scale video datasets. Finally, Section 5 serves as a wrap-up of our current contribution and provides an outlook of future work.

Notation We index an entry in a d-dimensional tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times p_2 \times \ldots \times p_d}$ using round parentheses such as $\boldsymbol{\mathcal{A}}(l_1, l_2, ..., l_d) \in \mathbb{R}$ and $\boldsymbol{\mathcal{A}}(l_1) \in \mathbb{R}^{p_2 \times p_3 \times \ldots \times p_d}$, when we only write the first index. Similarly, we also use $\boldsymbol{\mathcal{A}}(l_1, l_2) \in \mathbb{R}^{p_3 \times p_4 \times \ldots \times p_d}$ to refer to the sub-tensor specified by two indices l_1 and l_2.

2. Related Works

The current approaches to model video data are closely related to models for image data. A large majority of these works use deep CNNs to process each frame as image, and aggregate the CNN outputs. (Karpathy et al., 2014) proposes multiple fusion techniques such as Early, Late and Slow Fusions, covering different aspects of the video. This approach, however, does not fully take the order of frames into account. (Yue-Hei Ng et al., 2015) and (Fernando & Gould, 2016) apply global pooling of frame-wise CNNs, before feeding the aggregated information to the final classifier. An intuitive and appealing idea is to fuse these frame-wise spatial representations learned by CNNs using RNNs. The major challenge, however, is the computation complexity; and for this reason multiple compromises in the model design have to be made: (Srivastava et al., 2015) restricts the length of the sequences to be 16, while (Sharma et al., 2015) and (Donahue et al., 2015) use pre-trained CNNs. (Shi et al., 2015) proposed a more compact solution that applies convolutional layers as input-to-hidden and hidden-to-hidden mapping in LSTM. However, they did not show its performance on large-scale video data. (Simonyan & Zisserman, 2014) applied two stacked CNNs, one for spatial features and the other for temporal ones, and fused the outcomes of both using averaging and a Support-Vector Machine as classifier. This approach is further enhanced with Residual Networks in (Feichtenhofer et al., 2016). To the best of our knowledge, there has been no published work on applying pure RNN models to video classification or related tasks.

The Tensor-Train was first introduced by (Oseledets, 2011) as a tensor factorization model with the advantage of being capable of scaling to an arbitrary number of dimensions. (Novikov et al., 2015) showed that one could reshape a fully connected layer into a high-dimensional tensor and then factorize this tensor using Tensor-Train. This was applied to compress very large weight matrices in deep Neural Networks where the entire model was trained end-to-end. In these experiments they compressed fully connected layers on top of convolution layers, and also proved that a Tensor-Train Layer can directly consume pixels of image data such as CIFAR-10, achieving the best result among all known non-convolutional models. Then in (Garipov et al., 2016) it was shown that even the convolutional layers themselves can be compressed with Tensor-Train Layers. Actually, in an earlier work by (Lebedev et al., 2014) a similar approach had also been introduced, but their CP factorization is calculated in a pre-processing step and is only fine tuned with error back propagation as a post processing step.

(Koutnik et al., 2014) performed two sequence classification tasks using multiple RNN architectures of relatively low dimensionality: The first task was to classify spoken

words where the input sequence had a dimension of 13 channels. In the second task, RNNs were trained to classify handwriting based on the time-stamped 4D spatial features. RNNs have been also applied to classify the sentiment of a sentence such as in the IMDB reviews dataset (Maas et al., 2011). In this case, the word embeddings form the input to RNN models and they may have a dimension of a few hundreds. The sequence classification model can be seen as a special case of the Encoder-Decoder-Framework (Sutskever et al., 2014) in the sense that a classifier decodes the learned representation for the entire sequence into a probabilistic distribution over all classes.

3. Tensor-Train RNN

In this section, we first give an introduction to the core ingredient of our proposed approach, i.e., the Tensor-Train Factorization, and then use this to formulate a so-called Tensor-Train Layer (Novikov et al., 2015) which replaces the weight matrix mapping from the input vector to the hidden layer in RNN models. We emphasize that such a layer is learned end-to-end, together with the rest of the RNN in a very efficient way.

3.1. Tensor-Train Factorization

A *Tensor-Train Factorization* (TTF) is a tensor factorization model that can scale to an arbitrary number of dimensions. Assuming a d-dimensional target tensor of the form $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times p_2 \times ... \times p_d}$, it can be factorized in form of:

$$\widehat{\boldsymbol{\mathcal{A}}}(l_1, l_2, ..., l_d) \overset{TTF}{=} \boldsymbol{\mathcal{G}}_1(l_1)\, \boldsymbol{\mathcal{G}}_2(l_2)\, ...\, \boldsymbol{\mathcal{G}}_d(l_d) \tag{1}$$

where

$$\begin{aligned} &\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{p_k \times r_{k-1} \times r_k},\ l_k \in [1, p_k]\ \forall k \in [1, d] \\ &\text{and}\ \ r_0 = r_d = 1. \end{aligned} \tag{2}$$

As Eq. 1 suggests, each entry in the target tensor is represented as a sequence of matrix multiplications. The set of tensors $\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d$ are usually called core-tensors. The complexity of the TTF is determined by the ranks $[r_0, r_1, ..., r_d]$. We demonstrate this calculation also in Fig. 1. Please note that the dimensions and core-tensors are indexed from 1 to d while the rank index starts from 0; also note that the first and last ranks are both restricted to be 1, which implies that the first and last core tensors can be seen as matrices so that the outcome of the chain of multiplications in Eq. 1 is always a scalar.

If one imposes the constraint that each integer p_k as in Eq. (1) can be factorized as $p_k = m_k \cdot n_k\ \forall k \in [1, d]$, and consequently reshapes each $\boldsymbol{\mathcal{G}}_k$ into $\boldsymbol{\mathcal{G}}_k^* \in \mathbb{R}^{m_k \times n_k \times r_{k-1} \times r_k}$, then each index l_k in Eq. (1) and (2) can be uniquely represented with two indices (i_k, j_k), i.e.

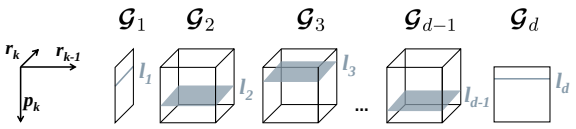

Figure 1: Tensor-Train Factorization Model: To reconstruct one entry in the target tensor, one performs a sequence of vector-matrix-vector multiplications, yielding a scalar.

$$i_k = \lfloor \frac{l_k}{n_k} \rfloor,\ j_k = l_k - n_k \lfloor \frac{l_k}{n_k} \rfloor, \tag{3}$$

$$\text{so that}\ \ \boldsymbol{\mathcal{G}}_k(l_k) = \boldsymbol{\mathcal{G}}_k^*(i_k, j_k) \in \mathbb{R}^{r_{k-1} \times r_k}. \tag{4}$$

Correspondingly, the factorization for the tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{(m_1 \cdot n_1) \times (m_2 \cdot n_2) \times ... \times (m_d \cdot n_d)}$ can be rewritten equivalently to Eq.(1):

$$\begin{aligned} &\widehat{\boldsymbol{\mathcal{A}}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \\ &\overset{TTF}{=} \boldsymbol{\mathcal{G}}_1^*(i_1, j_1)\, \boldsymbol{\mathcal{G}}_2^*(i_2, j_2)\, ...\, \boldsymbol{\mathcal{G}}_d^*(i_d, j_d). \end{aligned} \tag{5}$$

This double index trick (Novikov et al., 2015) enables the factorizing of weight matrices in a feed-forward layer as described next.

3.2. Tensor-Train Factorization of a Feed-Forward Layer

Here we factorize the weight matrix $\boldsymbol{W}$ of a fully-connected feed-forward layer denoted in $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.

First we rewrite this layer in an equivalent way with scalars as:

$$\begin{aligned} &\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{W}(i, j) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j) \\ &\forall j \in [1, N] \text{ and with } \boldsymbol{x} \in \mathbb{R}^M,\ \boldsymbol{y} \in \mathbb{R}^N. \end{aligned} \tag{6}$$

Then, if we assume that $M = \prod_{k=1}^d m_k$, $N = \prod_{k=1}^d n_k$ i.e. both M and N can be factorized into two integer arrays of the same length, then we can reshape the input vector $\boldsymbol{x}$ and the output vector $\hat{\boldsymbol{y}}$ into two tensors with the same number of dimensions: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$, and the mapping function $\mathbb{R}^{m_1 \times m_2 \times ... \times m_d} \rightarrow \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$ can be written as:

$$\begin{aligned} &\widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) \\ &= \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \cdot \\ &\quad \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d). \end{aligned} \tag{7}$$

Note that Eq. (6) can be seen as a special case of Eq. (7) with $d = 1$. The d-dimensional double-indexed tensor of weights $\mathcal{W}$ in Eq.(7) can be replaced by its TTF representation:

$$\begin{aligned}&\widehat{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d))\\ &\overset{TTF}{=} \mathcal{G}_1^*(i_1, j_1)\, \mathcal{G}_2^*(i_2, j_2)\, ...\, \mathcal{G}_d^*(i_d, j_d).\end{aligned} \tag{8}$$

Now instead of explicitly storing the full tensor $\mathcal{W}$ of size $\prod_{k=1}^d m_k \cdot n_k = M \cdot N$, we only store its TT-format, i.e., the set of low-rank core tensors $\{\mathcal{G}_k\}_{k=1}^d$ of size $\sum_{k=1}^d m_k \cdot n_k \cdot r_{k-1} \cdot r_k$, which can approximately reconstruct $\mathcal{W}$.

The forward pass complexity (Novikov et al., 2015) for one scalar in the output vector indexed by $(j_1, j_2, ..., j_d)$ turns out to be $\mathcal{O}(d \cdot \tilde{m} \cdot \tilde{r}^2)$. Since one needs an iteration through all such tuples, yielding $\mathcal{O}(\tilde{n}^d)$, the total complexity for one Feed-Forward-Pass can be expressed as $\mathcal{O}(d \cdot \tilde{m} \cdot \tilde{r}^2 \cdot \tilde{n}^d)$, where $\tilde{m} = \max_{k\in[1,d]} m_k, \tilde{n} = \max_{k\in[1,d]} n_k, \tilde{r} = \max_{k\in[1,d]} r_k$. This, however, would be $\mathcal{O}(M \cdot N)$ for a fully-connected layer.

One could also compute the compression rate as the ratio between the number of weights in a fully connected layer and that in its compressed form as:

$$r = \frac{\sum_{k=1}^d m_k n_k r_{k-1} r_k}{\prod_{k=1}^d m_k n_k}. \tag{9}$$

For instance, an RGB frame of size $160 \times 120 \times 3$ implies an input vector of length 57,600. With a hidden layer of size, say, 256 one would need a weight matrix consisting of 14,745,600 free parameters. On the other hand, a TTL that factorizes the input dimension with $8{\times}20{\times}20{\times}18$ is able to represent this matrix using 2,976 parameters with a TT-rank of 4, or 4,520 parameters with a TT-rank of 5 (Tab. 1), yielding compression rates of 2.0e-4 and 3.1e-4, respectively.

For the rest of the paper, we term a fully-connected layer in form of $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, whose weight matrix $\boldsymbol{W}$ is factorized with TTF, a *Tensor-Train Layer* (TTL) and use the notation

$$\hat{\boldsymbol{y}} = TTL(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}), \text{or } TTL(\boldsymbol{W}, \boldsymbol{x}) \tag{10}$$

where in the second case no bias is required. Please also note that, in contrast to (Lebedev et al., 2014) where the weight tensor is firstly factorized using non-linear Least-Square method and then fine-tuned with Back-Propagation, the TTL is always trained end-to-end. For details on the gradients calculations please refer to Section 5 in (Novikov et al., 2015).

3.3. Tensor-Train RNN

In this work we investigate the challenge of modeling high-dimensional sequential data with RNNs. For this reason, we factorize the matrix mapping from the input to the hidden layer with a TTL. For an Simple RNN (SRNN), which is also known as the Elman Network, this mapping is realized as a vector-matrix multiplication, whilst in case of LSTM and GRU, we consider the matrices that map from the input vector to the gating units:

TT-GRU:

$$\begin{aligned}\boldsymbol{r}^{[t]} &= \sigma(TTL(\boldsymbol{W}^r, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r)\\ \boldsymbol{z}^{[t]} &= \sigma(TTL(\boldsymbol{W}^z, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z)\\ \boldsymbol{d}^{[t]} &= \tanh(TTL(\boldsymbol{W}^d, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^d(\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]}))\\ \boldsymbol{h}^{[t]} &= (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]},\end{aligned} \tag{11}$$

TT-LSTM:

$$\begin{aligned}\boldsymbol{k}^{[t]} &= \sigma(TTL(\boldsymbol{W}^k, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k)\\ \boldsymbol{f}^{[t]} &= \sigma(TTL(\boldsymbol{W}^f, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f)\\ \boldsymbol{o}^{[t]} &= \sigma(TTL(\boldsymbol{W}^o, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o)\\ \boldsymbol{g}^{[t]} &= \tanh(TTL(\boldsymbol{W}^g, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g)\\ \boldsymbol{c}^{[t]} &= \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]}\\ \boldsymbol{h}^{[t]} &= \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}).\end{aligned} \tag{12}$$

One can see that LSTM and GRU require 4 and 3 TTLs, respectively, one for each of the gating units. Instead of calculating these TTLs successively (which we call vanilla TT-LSTM and vanilla TT-GRU), we increase n_1 —the first [1] of the factors that form the output size $N = \prod_{k=1}^d n_k$ in a TTL— by a factor of 4 or 3, and concatenate all the gates as one output tensor, thus parallelizing the computation. This trick, inspired by the implementation of standard LSTM and GRU in (Chollet, 2015), can further reduce the number of parameters, where the concatenation is actually participating in the tensorization. The compression rate for the input-to-hidden weight matrix $\boldsymbol{W}$ now becomes

$$r^* = \frac{\sum_{k=1}^d m_k n_k r_{k-1} r_k + (c-1)(m_1 n_1 r_0 r_1)}{c \cdot \prod_{k=1}^d m_k n_k} \tag{13}$$

where $c = 4$ in case of LSTM and 3 in case of GRU,

and one can show that r^* is always smaller than r as in Eq. 9. For the former numerical example of a input frame size $160{\times}120{\times}3$, a vanilla TT-LSTM would simply require 4 times as many parameters as a TTL, which would be 11,904 for rank 4 and 18,080 for rank 5. Applying this trick would, however, yield only 3,360 and 5,000 parameters for both ranks, respectively. We cover other possible settings of this numerical example in Tab. 1.

Finally to construct the classification model, we denote the i-th sequence of variable length T_i as a set of vectors

[1] Though in theory one could of course choose any n_k.

Table 1: A numerical example of compressing with TT-RNNs. Assuming that an input dimension of 160×120×3 is factorized as $8 \times 20 \times 20 \times 18$ and the hidden layer as $4 \times 4 \times 4 \times 4 = 256$, depending on the TT-ranks we calculate the number of parameters necessary for a Fully-Connected (FC) layer, a TTL which is equivalent to TT-SRNN , TT-LSTM and TT-GRU in their respective vanilla and parallelized form. For comparison, typical CNNs for preprocessing images such as AlexNet (Krizhevsky et al., 2012; Han et al., 2015) or GoogLeNet (Szegedy et al., 2015) consist of over 61 and 6 million parameters, respectively.

FC	TT-ranks	TTL	vanilla TT-LSTM	TT-LSTM	vanilla TT-GRU	TT-GRU
14,745,600	3	1,752	7,008	2,040	5,256	1,944
	4	2,976	11,904	3,360	8,928	3,232
	5	4,520	18,080	5,000	13,560	4,840

$\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$ with $\boldsymbol{x}_i^{[t]} \in \mathbb{R}^M \forall t$. For video data each $\boldsymbol{x}_i^{[t]}$ would be an RGB frame of 3 dimensions. For the sake of simplicity we denote an RNN model, either with or without TTL, with a function $f(\cdot)$:

$$\boldsymbol{h}_i^{[T_i]} = f(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}), \text{ where } \boldsymbol{h}_i^{[T_i]} \in \mathbb{R}^N, \tag{14}$$

which outputs the last hidden layer vector $\boldsymbol{h}_i^{[T_i]}$ out of a sequential input of variable length. This vector can be interpreted as a latent representation of the whole sequence, on top of which a parameterized classifier $\phi(\cdot)$ with either softmax or logistic activation produces the distribution over all J classes:

$$\begin{aligned}\mathbb{P}(\boldsymbol{y}_i = \mathbf{1}|\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}) &= \phi(\boldsymbol{h}_i^{[T_i]}) \\ &= \phi(f(\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})) \in [0,1]^J,\end{aligned} \tag{15}$$

The model is also illustrated in Fig. 2:

4. Experiments

In the following, we present our experiments conducted on three large video datasets. These empirical results demonstrate that the integration of the Tensor-Train Layer in plain RNN architectures such as a tensorized LSTM or GRU boosts the classification quality of these models tremendously when directly exposed to high-dimensional input data, such as video data. In addition, even though the plain architectures are of very simple nature and very low complexity opposed to the state-of-the-art solutions on these datasets, it turns out that the integration of the Tensor-Train Layer alone makes these simple networks very competitive to the state-of-the-art, reaching second best results in all cases.

UCF11 Data (Liu et al., 2009)

We first conduct experiments on the UCF11 – earlier known as the YouTube Action Dataset. It contains in total 1600 video clips belonging to 11 classes that summarize the human action visible in each video clip such as basketball shooting, biking, diving etc.. These videos originate from YouTube and have natural background ('in the wild') and a resolution of 320 × 240. We generate a sequence of RGB frames of size 160 × 120 from each clip at an fps(frame per second) of 24, corresponding to the standard value in film and television production. The lengths of frame sequences vary therefore between 204 to 1492 with an average of 483.7.

Figure 3: Two samples of frame sequences from the UCF11 dataset. The two rows belong to the classes of basketball shooting and volleyball spiking, respectively.

For both the TT-GRUs and TT-LSTMs the input dimension at each time step is $160 \times 120 \times 3 = 57600$ which is factorized as $8 \times 20 \times 20 \times 18$, the hidden layer is chosen to be $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are $[1, 4, 4, 4, 1]$. A fully-connected layer for such a mapping would have required 14,745,600 parameters to learn, while the input-to-hidden layer in TT-GRU and TT-LSTM consist of only 3,360 and 3,232, respectively.

As the first baseline model we sample 6 random frames in ascending order. The model is a simple Multilayer Perceptron (MLP) with two layers of weight matrices, the first of which being a TTL. The input is the concatenation of all 6 flattened frames and the hidden layer is of the same size as the hidden layer in TT-RNNs. We term this model as Tensor-Train Multilayer Perceptron (TT-MLP) for the rest of the paper. As the second baseline model we use plain GRUs and LSTMs that have the same size of hidden layer as their TT pendants. We follow (Liu et al., 2013) and perform for each experimental setting a 5-fold cross validation with mutual exclusive data splits. The mean and standard deviation of the prediction accuracy scores are reported in Tab. 2.

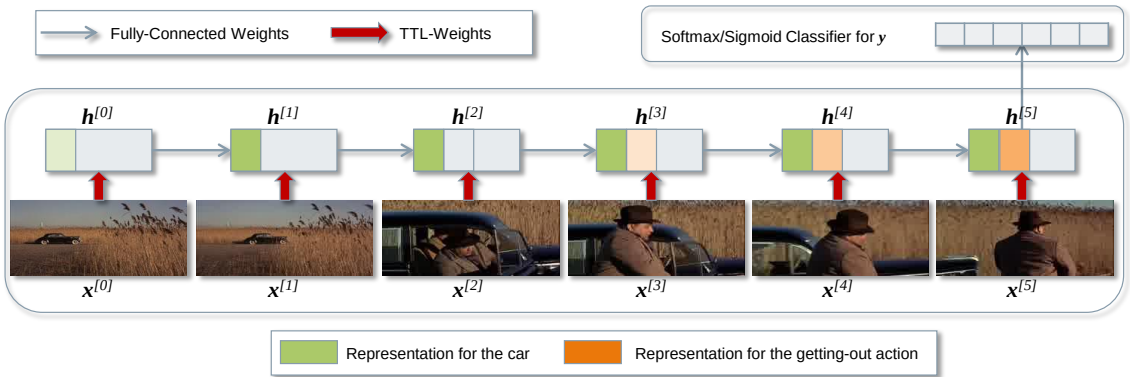

Figure 2: Architecture of the proposed model based on TT-RNN (For illustrative purposes we only show 6 frames): A softmax or sigmoid classifier built on the last hidden layer of a TT-RNN. We hypothesize that the RNN can be encouraged to aggregate the representations of different shots together and produce a global representation for the whole sequence.

Table 2: Experimental Results on UCF11 Dataset. We report i) the accuracy score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on a Quad core Intel®Xeon®E7-4850 v2 2.30GHz Processor to a maximum of 100 epochs

	Accuracy	# Parameters	Runtime
TT-MLP	0.427 ± 0.045	7,680	902s
GRU	0.488 ± 0.033	44,236,800	7,056s
LSTM	0.492 ± 0.026	58,982,400	8,892s
TT-GRU	$\mathbf{0.813 \pm 0.011}$	3,232	1,872s
TT-LSTM	0.796 ± 0.035	3,360	2,160s

The standard LSTM and GRU do not show large improvements compared with the TT-MLP model. The TT-LSTM and TT-GRU, however, do not only compress the weight matrix from over 40 millions to 3 thousands, but also significantly improve the classification accuracy. It seems that plain LSTM and GRU are not adequate to model such high-dimensional sequential data because of the large weight matrix from input to hidden layer. Compared to some latest state-of-the-art performances in Tab. 3, our model —simple as it is— shows accuracy scores second to (Sharma et al., 2015), which uses pre-trained GoogLeNet CNNs plus 3-fold stacked LSTM with attention mechanism. Please note that a GoogLeNet CNN alone consists of over 6 million parameters (Szegedy et al., 2015). In term of runtime, the plain GRU and LSTM took on average more than 8 and 10 days to train, respectively; while the TT-GRU and TT-LSTM both approximately 2 days. Therefore please note the TTL reduces the training time by a factor of 4 to 5 on these commodity hardwares.

Table 3: State-of-the-art results on the UCF11 Dataset, in comparison with our best model. Please note that there was an update of the data set on 31th December 2011. We therefore only consider works posterior to this date.

Original: (Liu et al., 2009)	0.712
(Liu et al., 2013)	0.761
(Hasan & Roy-Chowdhury, 2014)	0.690
(Sharma et al., 2015)	**0.850**
Our best model (TT-GRU)	0.813

Hollywood2 Data (Marszałek et al., 2009)

The Hollywood2 dataset contains video clips from 69 movies, from which 33 movies serve as training set and 36 movies as test set. From these movies 823 training clips and 884 test clips are generated and each clip is assigned one or multiple of 12 action labels such as answering the phone, driving a car, eating or fighting a person. This data set is much more realistic and challenging since the same action could be performed in totally different style in front of different background in different movies. Furthermore, there are often montages, camera movements and zooming within a single clip.

The original frame sizes of the videos vary, but based on the majority of the clips we generate frames of size 234×100, which corresponds to the Anamorphic Format, at fps of 12. The length of training sequences varies from 29 to 1079 with an average of 134.8; while the length of test sequences varies from 30 to 1496 frames with an average of 143.3.

The input dimension at each time step, being $234 \times 100 \times 3 = 70200$, is factorized as $10 \times 18 \times 13 \times 30$. The hidden layer is still $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are $[1, 4, 4, 4, 1]$. Since each clip might have more

Figure 4: Two samples of frame sequences from the Hollywood2 dataset. The first sequence (row 1 and 2) belongs to the class of sitting down; the second sequence (row 3 and 4) has two labels: running and fighting person.

than one label (multi-class multi-label problem) we implement a logistic activated classifier for each class on top of the last hidden layer. Following (Marszałek et al., 2009) we measure the performances using Mean Average Precision across all classes, which corresponds to the Area-Under-Precision-Recall-Curve.

As before we conduct experiments on this dataset using the plain LSTM, GRU and their respective TT modifications. The results are presented in in Tab. 4 and state-of-the-art in Tab. 5.

Table 4: Experimental Results on Hollywood2 Dataset. We report i) the Mean Average Precision score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on an NVIDIA Tesla K40c Processor to a maximum of 500 epochs.

	MAP	# Parameters	Runtime
TT-MLP	0.103	4,352	16s
GRU	0.249	53,913,600	106s
LSTM	0.108	71,884,800	179s
TT-GRU	0.537	2,944	96s
TT-LSTM	**0.546**	3,104	102s

(Fernando et al., 2015) and (Jain et al., 2013) use improved trajectory features with Fisher encoding (Wang & Schmid, 2013) and Histogram of Optical Flow (HOF) features (Laptev et al., 2008), respectively, and achieve so far the best score. (Sharma et al., 2015) and (Fernando & Gould, 2016) provide best scores achieved with Neural Network models but only the latter applies end-to-end training. To this end, the TT-LSTM model provides the second best score in general and the best score with Neural Network models, even though it merely replaces the input-to-hidden mapping with a TTL. Please note the large difference between the plain LSTM/GRU and the TT-LSTM/GRU, which highlights the significant performance improvements the Tensor-Train Layer contributes to the RNN models.

It is also to note that, although the plain LSTM and GRU consist of up to approximately 23K as many parameters as their TT modifications do, the training *time* does not reflect such discrepancy due to the good parallelization power of GPUs. However, the obvious difference in their training *qualities* confirms that training larger models may require larger amounts of data. In such cases, powerful hardwares are no guarantee for successful training.

Table 5: State-of-the-art Results on Hollywood2 Dataset, in comparison with our best model.

Original: (Marszałek et al., 2009)	0.326
(Le et al., 2011)	0.533
(Jain et al., 2013)	0.542
(Sharma et al., 2015)	0.439
(Fernando et al., 2015)	**0.720**
(Fernando & Gould, 2016)	0.406
Our best model (TT-LSTM)	0.546

Youtube Celebrities Face Data (Kim et al., 2008)
This dataset consists of 1910 Youtube video clips of 47 prominent individuals such as movie stars and politicians. In the simplest cases, where the face of the subject is visible as a long take, a mere frame level classification would suffice. The major challenge, however, is posed by the fact that some videos involve zooming and/or changing the angle of view. In such cases a single frame may not provide enough information for the classification task and we believe it is advantageous to apply RNN models that can aggregate frame level information over time.

Figure 5: Two samples of frame sequences from the Youtube Celebrities Face dataset. The two rows belong to the classes of Al Pacino and Emma Thompson.

The original frame sizes of the videos vary but based on the majority of the clips we generate frames of size 160 $\times$ 120 at fps of 12. The retrieved sequences have lengths varying from 2 to 85 with an average of 39.9. The input dimension at each time step is $160 \times 120 \times 3 = 57600$ which is factorized as $4 \times 20 \times 20 \times 36$, the hidden layer is again $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are

$[1, 4, 4, 4, 1]$.

Table 6: Experimental Results on Youtube Celebrities Face Dataset. We report i) the Accuracy score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on an NVIDIA Tesla K40c Processor to a maximum of 100 epochs.

	Accuracy	# Parameters	Runtime
TT-MLP	0.512 ± 0.057	3,520	14s
GRU	0.342 ± 0.023	38,880,000	212s
LSTM	0.332 ± 0.033	51,840,000	253s
TT-GRU	$\mathbf{0.800 \pm 0.018}$	3,328	72s
TT-LSTM	0.755 ± 0.033	3,392	81s

As expected, the baseline of TT-MLP model tends to perform well on the simpler video clips where the position of the face remains less changed over time, and can even outperform the plain GRU and LSTM. The TT-GRU and TT-LSTM, on the other hand, provide accuracy very close to the best state-of-the-art model (Tab. 7) using Mean Sequence Sparse Representation-based Classification (Ortiz et al., 2013) as feature extraction.

Table 7: State-of-the-art Results on Youtube Celebrities Face Dataset, in comparison with our best model.

Original: (Kim et al., 2008)	0.712
(Harandi et al., 2013)	0.739
(Ortiz et al., 2013)	**0.808**
(Faraki et al., 2016)	0.728
Our best model (TT-GRU)	0.800

Experimental Settings
We applied 0.25 Dropout (Srivastava et al., 2014) for both input-to-hidden and hidden-to-hidden mappings in plain GRU and LSTM as well as their respective TT modifications; and 0.01 ridge regularization for the single-layered classifier. The models were implemented in Theano (Bastien et al., 2012) and deployed in Keras (Chollet, 2015). We used the Adam (Kingma & Ba, 2014) step rule for the updates with an initial learning rate 0.001.

5. Conclusions and Future Work

We proposed to integrate Tensor-Train Layers into Recurrent Neural Network models including LSTM and GRU, which enables them to be trained end-to-end on high-dimensional sequential data. We tested such integration on three large-scale realistic video datasets. In comparison to the plain RNNs, which performed very poorly on these video datasets, we could empirically show that the integration of the Tensor-Train Layer alone significantly improves the modeling performances. In contrast to related works that heavily rely on deep and large CNNs, one advantage of our classification model is that it is simple and lightweight, reducing the number of free parameters from tens of millions to thousands. This would make it possible to train and deploy such models on commodity hardware and mobile devices. On the other hand, with significantly less free parameters, such tensorized models can be expected to be trained with much less labeled data, which are quite expensive in the video domain.

More importantly, we believe that our approach opens up a large number of possibilities to model high-dimensional sequential data such as videos using RNNs directly. In spite of its success in modeling other sequential data such as natural language, music data etc., RNNs have not been applied to video data in a fully end-to-end fashion, presumably due to the large input-to-hidden weight mapping. With TT-RNNs that can directly consume video clips on the pixel level, many RNN-based architectures that are successful in other applications, such as NLP, can be transferred to modeling video data: one could implement an RNN autoencoder that can learn video representations similar to (Srivastava et al., 2015), an Encoder-Decoder Network (Cho et al., 2014) that can generate captions for videos similar to (Donahue et al., 2015), or an attention-based model that can learn on which frame to allocate the attention in order to improve the classification.

We believe that the TT-RNN provides a fundamental building block that would enable the transfer of techniques from fields, where RNNs have been very successful, to fields that deal with very high-dimensional sequence data –where RNNs have failed in the past.

The source codes of our TT-RNN implementations and all the experiments in Sec. 4 are publicly available at `https://github.com/Tuyki/TT_RNN`. In addition, we also provide codes of unit tests, simulation studies as well as experiments performed on the HMDB51 dataset (Kuehne et al., 2011).

References

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Bingham, Ella and Mannila, Heikki. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250. ACM, 2001.

Cho, Kyunghyun, Van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chollet, François. Keras: Deep learning library for theano and tensorflow. https://github.com/fchollet/keras, 2015.

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

Faraki, Masoud, Harandi, Mehrtash T, and Porikli, Fatih. Image set classification by symmetric positive semi-definite matrices. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pp. 1–8. IEEE, 2016.

Feichtenhofer, Christoph, Pinz, Axel, and Wildes, Richard. Spatiotemporal residual networks for video action recognition. In *Advances in Neural Information Processing Systems*, pp. 3468–3476, 2016.

Fernando, Basura and Gould, Stephen. Learning end-to-end video classification with rank-pooling. In *Proc. of the International Conference on Machine Learning (ICML)*, 2016.

Fernando, Basura, Gavves, Efstratios, Oramas, Jose M, Ghodrati, Amir, and Tuytelaars, Tinne. Modeling video evolution for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5378–5387, 2015.

Garipov, Timur, Podoprikhin, Dmitry, Novikov, Alexander, and Vetrov, Dmitry. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Harandi, Mehrtash, Sanderson, Conrad, Shen, Chunhua, and Lovell, Brian C. Dictionary learning and sparse coding on grassmann manifolds: An extrinsic solution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3120–3127, 2013.

Hasan, Mahmudul and Roy-Chowdhury, Amit K. Incremental activity modeling and recognition in streaming videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 796–803, 2014.

Jain, Mihir, Jegou, Herve, and Bouthemy, Patrick. Better exploiting motion for better action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2555–2562, 2013.

Kambhatla, Nandakishore and Leen, Todd K. Dimension reduction by local principal component analysis. *Neural computation*, 9(7):1493–1516, 1997.

Karpathy, Andrej, Toderici, George, Shetty, Sanketh, Leung, Thomas, Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

Kim, Minyoung, Kumar, Sanjiv, Pavlovic, Vladimir, and Rowley, Henry. Face tracking and recognition with visual constraints in real-world videos. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008. URL http://seqam.rutgers.edu/site/index.php?option=com_content&view=article&id=64&Itemid=80.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Juergen. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

Laptev, Ivan, Marszalek, Marcin, Schmid, Cordelia, and Rozenfeld, Benjamin. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.

Le, Quoc V, Zou, Will Y, Yeung, Serena Y, and Ng, Andrew Y. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 3361–3368. IEEE, 2011.

Lebedev, Vadim, Ganin, Yaroslav, Rakhuba, Maksim, Oseledets, Ivan, and Lempitsky, Victor. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Liu, Dianting, Shyu, Mei-Ling, and Zhao, Guiru. Spatial-temporal motion information integration for action detection and recognition in non-static background. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pp. 626–633. IEEE, 2013.

Liu, Jingen, Luo, Jiebo, and Shah, Mubarak. Recognizing realistic actions from videos "in the wild". In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1996–2003. IEEE, 2009. URL http://crcv.ucf.edu/data/UCF_YouTube_Action.php.

Maas, Andrew L, Daly, Raymond E, Pham, Peter T, Huang, Dan, Ng, Andrew Y, and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Marszałek, Marcin, Laptev, Ivan, and Schmid, Cordelia. Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009. URL http://www.di.ens.fr/~laptev/actions/hollywood2/.

Novikov, Alexander, Podoprikhin, Dmitrii, Osokin, Anton, and Vetrov, Dmitry P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Ortiz, Enrique G, Wright, Alan, and Shah, Mubarak. Face recognition in movie trailers via mean sequence sparse representation-based classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3531–3538, 2013.

Oseledets, Ivan V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Sharma, Shikhar, Kiros, Ryan, and Salakhutdinov, Ruslan. Action recognition using visual attention. *arXiv preprint arXiv:1511.04119*, 2015.

Shi, Xingjian, Chen, Zhourong, Wang, Hao, Yeung, Dityan, Wong, Waikin, and Woo, Wangchun. Convolutional lstm network: a machine learning approach for precipitation nowcasting. pp. 802–810, 2015.

Simonyan, Karen and Zisserman, Andrew. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pp. 568–576, 2014.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Srivastava, Nitish, Mansimov, Elman, and Salakhutdinov, Ruslan. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Wang, Heng and Schmid, Cordelia. Action recognition with improved trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3551–3558, 2013.

Ye, Jieping, Janardan, Ravi, and Li, Qi. Gpca: an efficient dimension reduction scheme for image compression and retrieval. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 354–363. ACM, 2004.

Yue-Hei Ng, Joe, Hausknecht, Matthew, Vijayanarasimhan, Sudheendra, Vinyals, Oriol, Monga, Rajat, and Toderici, George. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4694–4702, 2015.

Zhang, Jun, Yan, Yong, and Lades, Martin. Face recognition: eigenface, elastic matching, and neural nets. *Proceedings of the IEEE*, 85(9):1423–1435, 1997.