TENSOR-TRAIN DECOMPOSITION FOR IMAGE RECOGNITION[*]

D. BRANDONI[†] AND V.SIMONCINI[‡]

Abstract. We explore the potential of Tensor-Train (TT) decompositions in the context of multi-feature face or object recognition strategies. We devise a new recognition algorithm that can handle three or more way tensors in the TT format, and propose a truncation strategy to limit memory usage. Numerical comparisons with other related methods – including the well established recognition algorithm based on high-order SVD – illustrate the features of the various strategies on benchmark datasets.

1. Introduction. Automatic Object Classification and Face Recognition have become an important component in activities such as security and video-surveillance, where the quantities under scrutiny are the image pixels; see, e.g., [19] for some applications. Several factors, including illumination, view angle and expression, can affect the image, making the process of classification more complex. In the past, numerical linear algebra tools, and primarily the Singular Value Decomposition (SVD) have been extensively used to automatically process images in computations for classification purposes, see, e.g., [9] and its references. In this context, a database of n_p persons in n_e expressions is stored as n_p distinct matrices $A_p \in \mathbb{R}^{n_i \times n_e}$, $p = 1, \dots, n_p$, where n_i is the number of pixels of each image. SVD-based classification algorithms work pretty well when the number of image pixels is greater than the number of features, otherwise the classification performance drops significantly. This suggests that alternative strategies should be used, since the latter condition often occurs in realistic applications where very many images need to be analyzed.

More recently, it has been shown that multilinear algebra and the algebra of higher-order tensors (see, e.g., [13]) can offer a more powerful mathematical framework for analyzing and addressing the multi-factor structure of image ensembles. For instance, the Weizmann database in [22] - where TensorFaces are introduced - is composed by 28 male subjects in 15 poses, 4 illumination conditions and 3 expressions, and it is represented by a 5-way tensor. Then the High-Order SVD (HOSVD, see [7] and its references) is used to classify the image of an unknown person. Other tensor-based approaches have been proposed in a large variety of contexts; for instance, in [12] Face Recognition is performed using tensor-tensor decompositions, while in [4] the classification problem is expressed using the Kronecker Product Equation (KPE) in a randomization context. The very recent applied literature in object classification and face recognition methods is quite vast and it now extends to various machine learning strategies such as neural network ([24],[8]) and deep learning ([2]) methodologies. For these reasons, it is hard to clearly capture the algorithmic properties and specific features of the proposed methods, and at the same time it is difficult to single out a winning method among those recently proposed. Here we would like to study (algebraic) matrix- and tensor-based approaches, whose performance can be analyzed in a well established manner. A qualitative comparison with a simple Neural Network architecture is also included; a deeper investigation of this comparison will be the subject of future research, due to the far more involved setting of the Neural

[*]This version dated January 8, 2020.

[†]Dipartimento di Matematica Università di Bologna, Piazza di Porta San Donato 5, Bologna, Italy (domitilla.brandoni2@unibo.it).

[‡]AM^2 and Dipartimento di Matematica Università di Bologna, Piazza di Porta San Donato 5, Bologna, and IMATI-CNR, Pavia (valeria.simoncini@unibo.it).

Network architecture. Indeed, the huge number of parameters to set and the strong dependency of the network performance on the database make the process of selecting a winning neural architecture extremely complex. Furthermore, there may be dramatic differences between the computational costs (in terms of CPU time) associated with training a Neural Network and training a tensor/matrix-based algorithm, with the latter possibly being orders of magnitude cheaper. On the other hand, if one can afford using a high number of layers and neurons, Neural Networks indeed perform well with a broad variety of databases.

We explore the use of the *Tensor-Train Decomposition* (TT-Decomposition) for multi-feature recognition strategies. This tensor decomposition has been shown to have great potential in multivariate function approximation and compression, and it is particularly amenable to truncation strategies yielding low-rank tensor approximations. The decomposition unwraps the given multi-way tensor in an appropriate chain ("train") of three-dimensional tensors and it is thus appropriate to handle high order tensors, as opposed to HOSVD, which requires to allocate memory for a core tensor with the same order of $\mathcal{A}$. We have thus devised a recognition algorithm based on the TT Decomposition, which extends the recognition method of HOSVD to this setting; to the best of our knowledge this algorithm is new.

A synopsis of the paper is as follows. In section 2 we recall some standard properties of tensor computations, that will be used throughout the paper. In section 3 we present the TT-form of the given image-related tensor. With section 4 we start our discussion of classification algorithms, beginning with a simple minded least-squares method, and then relating it to the algorithms based on the HOSVD. In subsection 4.3 we introduce our TT classification algorithm for the 3D case, making some algebraic comparisons with known methods and discussing a truncation strategy. In section 5 we extend our algorithm to fourth (or higher) order tensors. In section 6 we present the eight databases, which will be used to exercise the various algorithms in section 7 using different performance measures. We include our conclusions in section 8.

2. Tensor Computations.

In this section we summarize some of the basic facts about tensors and their computations, that will be used in the paper.

A higher-order tensor is a multidimensional array, corresponding to an element of a tensor product of N vector spaces. First-order tensors (that is, vectors) are usually denoted by lowercase letters (a, b, c, ...), second-order tensors (matrices) are denoted by capital letters (A, B, C, ...) and higher-order tensors are denoted by calligraphic letters ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ...). The *order* of a tensor is the number of dimensions, also known as *ways* or *modes*. For example, a tensor $\mathcal{A} \in \mathbb{R}^{4\times 2\times 3}$ is a third-order tensor.

Two important tensor operations need to be introduced: the *n-mode product*, which is a tensor by matrix operation, and the $\binom{m}{n}$*-product*, which is a tensor-by-tensor operation.

Definition 2.1. [13, p. 460] *Given the tensor $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, the n-mode matrix product of $\mathcal{A}$ with a matrix $U \in \mathbb{R}^{J\times I_n}$ is denoted by*

$$\mathcal{A} \times_n U \in \mathbb{R}^{I_1\times\cdots\times I_{n-1}\times J\times I_{n+1}\times\cdots\times I_N}$$

and its entries are given by

$$(\mathcal{A} \times_n U)_{i_1,\dots,i_{n-1},j,i_{n+1},\dots,I_N} = \sum_{i_n=1}^{I_n} a_{i_1,i_2,\dots,i_n,\dots,i_N} u_{j,i_n}$$

The n-mode multiplication satisfies the following important commutativity property,

$$\mathcal{A} \times_n U \times_m V = \mathcal{A} \times_m V \times_n U \qquad \forall\, m \neq n,$$

where $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_m \times \cdots \times I_N}$, $U \in \mathbb{R}^{J \times I_n}$ and $V \in \mathbb{R}^{J \times I_m}$. Therefore, in a series of multiplications the product order of distinct modes is irrelevant. If the modes are the same, i.e. $m = n$, then

$$\mathcal{A} \times_n U \times_n V = \mathcal{A} \times_n (VU).$$

In a similar way to the n-mode product, in [6] the $\binom{m}{n}$-product of a tensor is introduced.

DEFINITION 2.2. [6] *The $\binom{m}{n}$-product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a tensor $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$, such that $I_n = J_m$, is defined as*

$$\mathcal{C} = \mathcal{A} \times_n^m \mathcal{B},$$

where $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$. Its entries are given by

$$\mathcal{C}(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N, j_1, \ldots j_{m-1}, j_{m+1}, \ldots, j_M) = \sum_{i=1}^{I_n} \mathcal{A}(i_1, \ldots, i_{n-1}, i, i_{n+1}, \ldots, i_N) \mathcal{B}(j_1, \ldots, j_{m-1}, i, j_{m+1}, \ldots, j_M).$$

Finally, we define the process of flattening a tensor into a matrix, i.e. the *unfolding*. There are several ways to unfold a tensor into a matrix. In this work we refer to the one used by Oseledets in [15]. Let $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be an Nth-order tensor. The $k-th$ unfolding of $\mathcal{A}$ is a matrix $A_k \in \mathbb{R}^{(I_1 I_2 \ldots I_k) \times (I_{k+1} I_{k+2} \ldots I_N)}$, whose elements are taken column-wise from $\mathcal{A}$, that is

$$A_k(i_1 \cdot \ldots \cdot i_k, i_{k+1} \cdot \ldots \cdot i_N) = \mathcal{A}(i_1, \ldots, i_k, i_{k+1}, \ldots i_N). \tag{2.1}$$

3. Tensor-Train Decomposition. The Tensor-Train Decomposition factorizes an Nth-order tensor $\mathcal{A}$ in a product of third-order tensors and it is given by ([15])

$$\begin{aligned} \mathcal{A}(i_1, \ldots, i_N) &= G_1(i_1, :)\mathcal{G}_2(:, i_2, :)\mathcal{G}_3(:, i_3, :) \ldots G_N(i_N, :) \\ &= G_1(i_1)\mathcal{G}_2(i_2)\mathcal{G}_3(i_3) \ldots G_N(i_N), \end{aligned} \tag{3.1}$$

where G_1, $\mathcal{G}_2$, ..., G_N are called *TT-cores*.

In index form, the definition (3.1) can be written as

$$\mathcal{A}(i_1, \ldots, i_N) = \sum_{\alpha_1, \ldots, \alpha_{N-1}} G_1(i_1, \alpha_1)\mathcal{G}_2(\alpha_1, i_2, \alpha_2) \ldots G_N(\alpha_{N-1}, i_N). \tag{3.2}$$

A classical visualization of the decomposition of a third-order tensor in index form is given in Figure 3.1 (see [15]). In this work we consider the TT-SVD decomposition, i.e. each factor of the Tensor-Train Decomposition is computed using the SVD of a specific unfolding of $\mathcal{A}$. For example, the first term of the decomposition is obtained by considering the SVD of $A_1 = G_1 \Sigma_1 V_1^T$. For further details we refer the reader to the seminal work [15] and to [21].

Using Definition 2.2 for the $\binom{m}{n}$-mode product, (3.2) can be written as

$$\mathcal{A} = G_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3 \times_3^1 \cdots \times_3^1 G_N. \tag{3.3}$$

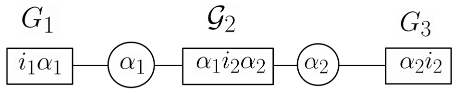

Fig. 3.1: Visualization of the Tensor Train Decomposition in index form.

Thus, if $\mathcal{A}$ is a third-order tensor, then

$$\mathcal{A} = \mathcal{G}_2 \times_1 G_1 \times_3 G_3^T, \tag{3.4}$$

since

$$\mathcal{A}(i_1, i_2, i_3) = G_1(i_1, :) \times_2^1 \mathcal{G}_2(:, i_2, :) \times_3^1 G_3(:, i_3) = \sum_{\alpha_1, \alpha_2} G_1(i_1, \alpha_1)\mathcal{G}_2(\alpha_1, i_2, \alpha_2)G_3(\alpha_2, i_3)$$
$$= \sum_{\alpha_2} (\mathcal{G}_2 \times_1 G_1)(i_1, i_2, \alpha_2)G_3(\alpha_2, i_3) = (\mathcal{G}_2 \times_1 G_1 \times_3 G_3^T)(i_1, i_2, i_3).$$

A standard visualization of (3.4) is given in Figure 3.2.

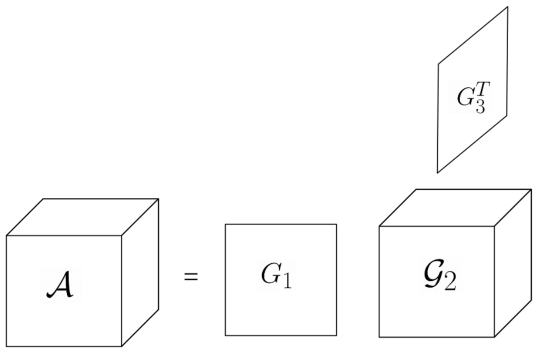

Fig. 3.2: Visualization of the Tensor Train Decomposition of a third-order tensor using the $\binom{m}{n}$-mode product.

We notice that for a third order tensor the Tensor-Train Decomposition corresponds to the classical Tucker2 decomposition (see, e.g., [17]), while HOSVD corresponds to a Tucker3 decomposition, when the involved matrices are orthogonal. Thorough overviews of various Tucker decompositions for a third-order tensor can be found for instance in [5],[11].

To make tensor computations affordable, rank truncation strategies are commonly employed; we refer the reader to, e.g., [12],[1],[16] for a discussion of various alternatives, to comply with the difficulties associated with the higher order setting.

In TT decompositions a crucial aspect is how to determine the TT-ranks $(R_1, \cdots, R_N)$, where each R_i is such that $\mathcal{G}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}$. To specify its TT-ranks, we have considered both a full representation, and a truncated version of the tensor. In the first case, R_i equals the number of columns of the matrix U_i, which contains the left singular vectors of the i-th unfolding of $\mathcal{A}$. Thus, no truncation is performed in the SVD

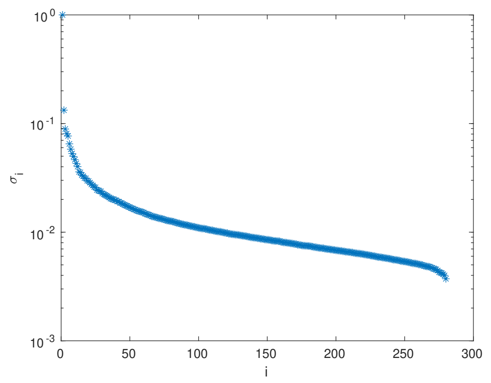

Fig. 3.3: Singular values of A_1 for the Orl Database.

along different modes. In the truncated version, we truncate the matrix U_i to its first k columns, according to a threshold π, where k satisfies

$$k = \max_J \left\{ \frac{\sum_{j=1}^{J} \sigma_j(A_i)}{\sum_{j=1}^{I_i} \sigma_j(A_i)} \leq \pi \right\},$$

where π is such that $k \geq 1$. This selection is in agreement with the truncation guidelines discussed in [1],[12],[14]. In our strategy we truncate only along the first mode, although the same procedure can be used to perform the truncation also along the second and third modes. Thus, we consider only the first k columns of G_1, and name the resulting matrix $\hat{G}_1$. As a consequence, we also set $\hat{\mathcal{G}}_2 \in \mathbb{R}^{k \times n_e \times n_p}$ as the tensor of the first k first-mode elements of $\mathcal{G}_2$. Hence the truncated TT-SVD decomposition has the following form

$$\mathcal{A} = \hat{\mathcal{G}}_2 \times_1 \hat{G}_1 \times_3 G_3^T.$$

The error due to this truncation can be estimated as described in [16, Th.2.2].

4. 3D Classification problem and algorithms.

In this and the next sections we recall some known tensor-based classification algorithms, and we introduce a classification strategy associated with the Tensor-Train decomposition. In this first section we consider the TT decomposition of a 3D tensor, while we postpone to later sections that of higher order tensors, anticipating that the TT formulation allows us to treat the two cases in exactly the same manner. To the best of our knowledge, the use of these strategies in the classification context is new.

Let $\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}$ be the tensor representing the database of images with n_i pixels, n_p persons and n_e expressions for each person. Given an image z of an unknown person in an unknown expression, represented by a vector in $\mathbb{R}^{n_i}$, we want to determine which of the n_p persons, the new image is closest to. To this end we define a distance $dist(z, \mathcal{A}(:,:,p))$ of z from each person p of the given database, for $p = 1, \ldots, n_p$.

In the following discussion $\mathbf{E}^e$ denotes the space "*expression* e". This space is spanned by all images of the n_p persons in expression e.

4.1. Least Squares Classification Algorithm.

This simple minded classification strategy merely focuses on the matrices obtained by comparing with respect

Algorithm 4.1 Least Squares

```
1: **Input**: $z \in \mathbb{R}^{n_i}$ input image and $\mathcal{A}$.
2: **for** $e = 1, \cdots, n_e$ **do**
3:     Solve for $x_e$ in (4.1) and set $r_e = z - A^e x_e$ [1]
4: **end for**
5: $\hat{e} = \underset{e}{\text{argmin}}(\|r_e\|)$
6: $\hat{p} = \underset{p}{\text{argmax}}|x_{\hat{e}}(p)|$
7: **Output**: Classify $z$ as person $\hat{p}$
```

to all images in each given expression. More precisely, for each expression e let $A^e := \mathcal{A}(:, e, :)$, where the columns of A^e have been scaled to have unit Euclidean norm. Consider the following least squares problem

$$\min_x \|z - A^e x\|_2. \tag{4.1}$$

Let x_e be the solution to (4.1), and let $r_e = z - A^e x_e$ be the associated residual. The distance of z from $\mathbf{E}^e$ is obtained by $\|r_e\|_2$. We then compute

$$\hat{e} = \text{argmin}_e \|r_e\|_2,$$

and we classify z as person $\hat{p}$, where $\hat{p} = \text{argmax}_p |x_{\hat{e}}(p)|$, that is $\hat{p}$ corresponds to the largest component of the $x_{\hat{e}}$, where $x_{\hat{e}}$ holds the linear combination of all persons in the closest expression. A version of the classification algorithm is given in Algorithm 4.1.

4.2. HOSVD Classification Algorithm. Using the HOSVD described in [7], the tensor $\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}$ can be written as follows

$$\mathcal{A} = \mathcal{S} \times_i F \times_e G \times_p H, \tag{4.2}$$

where $\times_i$, $\times_e$, $\times_p$ are the 1-mode, 2-mode, 3-mode multiplications, respectively[2]; with $G \in \mathbb{R}^{n_e \times n_e}$, $H \in \mathbb{R}^{n_p \times n_p}$ and, if $n_i > n_e n_p$ then $\mathcal{S} \in \mathbb{R}^{n_e n_p \times n_e \times n_p}$, $F \in \mathbb{R}^{n_i \times n_e n_p}$, otherwise, $\mathcal{S} \in \mathbb{R}^{n_i \times n_e \times n_p}$, $F \in \mathbb{R}^{n_i \times n_i}$.

Letting $\mathcal{C} = \mathcal{S} \times_i F \times_e G$, for a fixed expression e

$$A^e = C^e H^T \qquad e = 1, 2, \ldots, n_e, \tag{4.3}$$

where A^e and C^e are the matrices obtained by fixing the second index of $\mathcal{A}$ and $\mathcal{C}$ equal to e. Note that the $n_p \times n_p$ matrix H^T is orthogonal and that the matrix C^e is $n_i \times n_p$. Column p of A^e can be written as

$$a_p^{(e)} = C^e h_p^T, \tag{4.4}$$

where h_p^T is the p-th column of H^T. Now let $z \in \mathbb{R}^{n_i}$ be the new image to be classified. The coordinates of z in the expression basis can be found by solving a least squares problem

$$\hat{\alpha}_e = \text{argmin}_{\alpha_e} \|C^e \alpha_e - z\|_2, \qquad e = 1, \cdots, n_e. \tag{4.5}$$

[1] Each slice of $\mathcal{A}$, that is A^e, is normalized before solving (4.1).

[2] $\times_i$ is the *image-mode* multiplication, $\times_e$ is the *expression-mode* multiplication, $\times_p$ is the *person-mode* multiplication.

Then, for each $e = 1, \cdots, n_e$ and for each $p = 1, \cdots, n_p$

$$D_{HO}(e,p) = \|\hat{\alpha_e} - h_p^T\|_2.$$

Note that if C^e is full column rank, then the least squares problem (4.5) has a unique solution, otherwise the solution is not unique, but the solution with smallest norm can be obtained by using the SVD of C^e for its computation ([10, section 5.5]). A version of the classification algorithm based on the HOSVD can be found in [9, Chapter 14, p.174], and this is what was used in our experiments.

There is a tight connection between the LS-based method and the HOSVD classification, given that for each expression e the matrix A^e is the same.

Proposition 4.1. *Let x_e be the solution to (4.1) in the LS-based method, and let $\hat{\alpha}_e$ be the solution to (4.5) in the HOSVD method. Then $\hat{\alpha}_e = H^T x_e$. Letting $I_{:,p}$ be the p-th column of the canonical basis, it then holds*

$$D_{HO}(e,p) = \|\hat{\alpha_e} - h_p^T\|_2 = \|x_e - I_{:,p}\|_2.$$

Proof. The least squares solution $\hat{\alpha}_e$ is given by $\hat{\alpha}_e = ((C_e)^T C_e)^{-1}(C_e)^T z$, while the least squares solution x_e is given by $x_e = ((A^e)^T A^e)^{-1}(A^e)^T z$. Substituting $A^e = C^e H^T$ we obtain

$$\begin{aligned} x_e &= ((A^e)^T A^e)^{-1}(A^e)^T z = (H(C^e)^T C^e H^T)^{-1} H (C^e)^T z \\ &= H^{-T}((C^e)^T C^e)^{-1}(C^e)^T z = H^{-T}\hat{\alpha}_e, \end{aligned}$$

which proves the first assertion. For the second assertion, since $\hat{\alpha}_e = H^T x_e$, we have $\|\hat{\alpha_e} - h_p^T\|_2 = \|H^T x_e - H^T I_{:,p}\|_2 = \|H^T(x_e - I_{:,p})\|_2$. From the orthogonality of H the result follows. ∎

Assume that z and A^e have both been scaled to have unit norm columns, so that the entries of x_e are all not greater than one in absolute value. It follows that for a fixed e, finding the minimum of $D_{HO}(e,p)$ corresponds to finding the largest entry of x_e, the $\hat{p}$-th entry, which is closest to one. Hence, the difference between the two methods lays in the way the "best" expression is chosen. In the basic least squares method the expression corresponding to the smallest residual norm is selected, whereas in HOSVD all is based on the quantity $\|x_e - I_{:,p}\|$. This implies that the HOSVD and LS methods compute the same quantities, but the stopping criterion changes, which is thus responsible of possible discrepancies in the performance of the two approaches (see, e.g., the results for the Fashion MNIST dataset).

Table 4.1 displays the differences and similarities between the least-squares and the HOSVD formulations for the classification of a single image: the test set is composed by the person p in the last expression, while all the other n_p persons in the remaining expressions are used as training set. We report the classification result of each method by displaying $\hat{p}$ for each database. The method LS2 corresponds to Algorithm 4.1, in which lines 5 and 6 are replaced with $[\hat{e},\hat{p}] = \text{argmin} D(e,p) = \text{argmin}\|x_e - I_{:,p}\|$. According to Proposition 4.1, the same classification results are obtained using either HOSVD or LS2.

Another classification algorithm based on HOSVD is presented in [18], where the training set is decomposed as in (4.2). The tensor $\mathcal{L} = \mathcal{S}(1:r, 1:s, :) \times_3 H$ is then computed[3], and for every person p the first k singular vectors of $L^p = \mathcal{L}(:,:,p)$ are

[3]The parameters r, s can be set arbitrarily and depend on the desired data compression.

Database	p	LS	LS2	HOSVD
Orl	14	18	14	14
COIL-20	1	1	1	1
Faces95	5	50	57	57
Faces96	59	77	59	59
Ext'd Yale shrunk	9	10	6	6

Table 4.1: Classification performance of least squares based algorithms and the algorithm based on HOSVD.

stored in a matrix B^p. Let $z \in \mathbb{R}^{n_i}$ be the image of an unknown person in an unknown expression to be classified. The following least squares problem measures how far z is from being a linear combination of the columns of B^p for each $p = 1, \cdots, n_p$:

$$D(p) := \min_x \|z_p - B^p x\|_2, \qquad \text{where} \quad z_p = F(1:r,:)^T z. \tag{4.6}$$

Therefore, z is classified as person $\hat{p} = \text{argmin}_p(D)$. A more complete analysis of this method can be found in [18]. In Section 7 we will refer to this algorithm as HOSVD2. We conclude this brief description by noticing that HOSVD2 requires the computation of an HOSVD of the training tensor and n_p SVDs to determine the matrices B^p, $p = 1, \cdots, n_p$. This can be computationally expensive when n_p is large.

4.3. Tensor-Train Classification Algorithm. To derive the new classification algorithm we first write the data tensor for each expression e, by means of a TT-based basis for $\mathbf{E}^e$. Then we define the associated distance to be minimized.

Using (3.4), $\mathcal{A}(:,e,:)$ can be written as

$$\mathcal{A}(:,e,:) = G_2^e \times_i G_1 \times_p G_3^T \qquad \forall e = 1, \ldots, n_e, \tag{4.7}$$

where $G_2^e = \mathcal{G}_2(:,e,:)$ is a matrix. Thus, the image of a person p in the expression e is

$$\mathcal{A}(:,e,p) = G_1 G_2^e g_3^p, \qquad \text{where} \quad g_3^p = G_3(:,p). \tag{4.8}$$

To deal with the classification problem, it is better to set $\mathcal{C} = \mathcal{G}_2 \times_i G_1$ so that (4.8) becomes

$$\mathcal{A}(:,e,p) = C^e g_3^p, \qquad \text{with} \quad C^e = \mathcal{C}(:,e,:) = G_1 G_2^e. \tag{4.9}$$

This can be interpreted as follows. The columns of C^e are a basis for $\mathbf{E}^e$ while g_3^p, the p-th column of G_3, holds the coordinates of person p in this basis. Notice that the same g_3^p holds the coordinates of all the images of person p in the different expression bases [9, p.173]. The expression above plays the same role as (4.4) for the HOSVD.

Given the image z of an unknown person in an unknown expression, we want to find the coordinates α_e of z in all the n_e bases $\{C^e\}_{e=1,\ldots,n_e}$ and then compare each α_e for $e = 1, \ldots, n_e$ with the coordinates of all n_p persons in the same basis, which are represented by the columns of G_3. More precisely, for each $e = 1, \cdots, n_e$ we compute

$$\min_{\alpha_e} \|C^e \alpha_e - z\|_2, \tag{4.10}$$

and then, for each $e = 1, \cdots, n_e$ and for each $p = 1, \cdots, n_p$

$$D_{TT}(e,p) := \|\hat{\alpha_e} - g_3^p\|_2. \tag{4.11}$$

Hence, the distance between z and person p is given by

$$dist(z, \mathcal{A}(:,:,p)) = \min_e D_{TT}(e,p).$$

The computation in (4.10) can be performed by means of the reduced QR decomposition of G_2^e, that is $G_2^e = Q^e R^e$. The coordinates of z in $\mathbf{E}^e$ are thus given by $\hat{\alpha_e} = {R^{(e)}}^{-1} {Q^{(e)}}^T G_1^T z$ and so (4.11) becomes

$$D_{TT}(e,p) = \| {R^{(e)}}^{-1} {Q^{(e)}}^T G_1^T z - g_3^p \|_2 \qquad p = 1, \cdots, n_p,$$

where $g_3^p = G_3(:,p)$. The overall procedure is summarized in Algorithm 4.2.

Algorithm 4.2 Algorithm TT3D

```
**Input**: $z$ test image $G_1$, $\mathcal{G}_2$, $G_3$.
Compute $\hat{z} = G_1^T z$:
**for** $e = 1, \dots, n_e$ **do**
  Solve $G_2^e \alpha_e = \hat{z}$ for $\alpha_e$
  **for** $p = 1, \dots, n_p$ **do**
    $D_{TT}(e,p) = \|\alpha_e - g_3^p\|_2$ where $g_3^p = G_3(:,p)$.
  **end for**
**end for**
$(\hat{e}, \hat{p}) = \underset{e,p}{\text{argmin}}(D_{TT})$
**Output**: Classify $z$ as person $\hat{p}$
```

Remark 4.2. *Let $A^e = \mathcal{A}(:,e,:)$ in (4.8). Then a result analogous to that in Proposition 4.1 can be derived, that is the distance D_{TT} satisfies*

$$D_{TT}(e,p) = \|x_e - I_{:,p}\|$$

where $x_e = \text{argmin}_x \|A^e x - z\|$.

5. 4D Classification algorithm with Tensor-Train decomposition. Several authors in the literature have observed that image classification can be improved by considering images as matrices $I \in \mathbb{R}^{n_1 \times n_2}$ instead of vectors, thus adding a forth dimension to the problem; see, e.g., [12]. Hence, a database of n_p persons in n_e expressions is represented by a four-dimensional tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_e \times n_p}$, written in TT-form as

$$\mathcal{A} = G_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3 \times_3^1 G_4,$$

where $G_1 \in \mathbb{R}^{n_1 \times n_1}$, $\mathcal{G}_2 \in \mathbb{R}^{n_1 \times n_2 \times n_e n_p}$, $\mathcal{G}_3 \in \mathbb{R}^{n_e n_p \times n_e \times n_p}$, and $G_4 \in \mathbb{R}^{n_p \times n_p}$. Thus, the image of a person p in expression e in the database is given by

$$\mathcal{A}(:,:,e,p) = \mathcal{G}_2 \times_1 G_1 \times_3 (G_3^{(e)})^T \times_3 g_4^{(p)}, \tag{5.1}$$

where $G_3^{(e)} = \mathcal{G}_3(:,e,:) \in \mathbb{R}^{n_e n_p \times n_p}$ and $g_4^{(p)} = G_4(:,p) \in \mathbb{R}^{n_p}$. Let $\mathcal{C}^{(e)} = \mathcal{G}_2 \times_1 G_1 \times_3 (G_3^{(e)})^T$. Then (5.1) becomes

$$\mathcal{A}(:,:,e,p) = \mathcal{C}^{(e)} \times_3 g_4^{(p)}.$$

The classification strategy is then analogous to that for the three-dimensional case. In particular, given an image $z \in \mathbb{R}^{n_1 n_2}$ of an unknown person in an unknown expression, we define the distance of z from person p as

$$D_{TT4}(e,p) = \|\hat{\alpha_e} - g_4^{(p)}\|_2,$$

where

$$\hat{\alpha_e} = \text{argmin}_{\alpha_e} \|\text{unfold}(\mathcal{C}^{(e)})\alpha_e - z\|_2.$$

A version of the classification algorithm is given in Algorithm TT 4D. In our experiments, we used the unfolding `tenmat`($\mathcal{C}^e$,3,'t') from the Matlab Tensor Toolbox [3].

Algorithm TT 4D

```
Input: $z$ test image, $G_1$, $\mathcal{G}_2$, $\mathcal{G}_3, G_4$.
Compute $\mathcal{G}_{12} = \mathcal{G}_2 \times_1 G_1$:
for $e = 1, \ldots, n_e$ do
  Compute $\mathcal{C}^{(e)} = \mathcal{G}_{12} \times_3 (G_3^{(e)})^T$, where $G_3^{(e)} = \mathcal{G}_3(:,e,:)$
  Solve $\min_{\alpha_e} \|\text{unfold}(\mathcal{C}^{(e)})\alpha_e - z\|_2$ for $\alpha_e$
  for $p = 1, \ldots, n_p$ do
    $D_{TT4}(e,p) = \|\alpha_e - g_4^p\|_2$ where $g_4^p = G_4(:,p)$.
  end for
end for
$(\hat{e}, \hat{p}) = \underset{e,p}{\text{argmin}}(D_{TT4})$
Output: Classify $z$ as person $\hat{p}$
```

The TT format naturally extends to the higher-dimensional setting. For instance, suppose that $\mathcal{A}$ is an Nth order tensor representing our database, further assume that the first two modes are the pixel modes (as in TT4D) and the last one is the person mode. According to (3.3) $\mathcal{A}$ can be written in the following way:

$$\mathcal{A} = G_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3 \times_3^1 \cdots \times_3^1 \mathcal{G}_{N-1} \times_3^1 G_N.$$

Thus, person p in a specific combination of all the other $(N-3)$ features can be expressed as

$$\mathcal{A}(:,:,i_3,\cdots,i_{N-1},p) = G_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3^{(i_3)} \times_3^1 \cdots \times_3^1 \mathcal{G}_{N-1}^{(i_{N-1})} \times_3^1 G_N(:,p). \tag{5.2}$$

Let $C^{(i_3,\cdots,i_{N-1})} = G_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3^{(i_3)} \times_3^1 \cdots \times_3^1 \mathcal{G}_{N-1}^{(i_{N-1})}$. Then, (5.2) becomes

$$\mathcal{A}(:,:,i_3,\cdots,i_{N-1},p) = C^{(i_3,\cdots,i_{N-1})} \times_3 g_N^{(p)}, \tag{5.3}$$

where $g_N^{(p)} = G_N(:,p)$.

Given an image vector $z \in \mathbb{R}^{n_1 n_2}$ of an unknown person in an unknown combination of the $(N-3)$ features, we define the distance between z and the person p as

$$D_{TTN}(i_3,\cdots,i_{N-1},p) = \|\hat{\alpha}_{i_3,\cdots,i_{N-1}} - g_N^{(p)}\|,$$

where

$$\hat{\alpha}_{i_3,\cdots,i_{N-1}} = \operatorname{argmin}_{\alpha_{i_3,\cdots,i_{N-1}}} \|\text{unfold}(C^{(i_3,\cdots,i_{N-1})})\alpha_{i_3,\cdots,i_{N-1}} - z\|.$$

6. Description of the Databases. In the following we consider a database of faces of n_p persons in n_e different expressions – where by *expression* we possibly mean different illuminations, view angle, etc. – with $n_i = n_1 n_2$ pixels. Each image can be either considered as an $n_1 \times n_2$ matrix, or as a $n_1 n_2$ vector, so that such a database can be represented either by a 4-way tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_e \times n_p}$ or by a 3-way tensor $\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}$.

In this work we consider eight different datasets. For the first four ones it holds that $n_i < n_e$, while for the others $n_i > n_e$. This selection was performed to computationally illustrate the need to use tensor strategies whenever the number of pixel is larger than the number of expressions, to be able to classify a new image with a high success rate.

1. The **Orl database**, which contains 400 grayscale images in PGM format of 40 persons. Each subject is photographed in 10 different expressions. In Figure 6.1 all the expressions of subject 1 are shown.

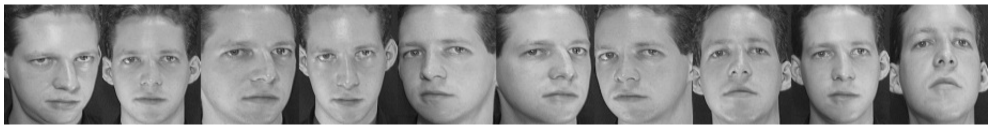

Fig. 6.1: Subject 1 of the Orl database.

2. The **Faces95 database**, which consists of 1440 RGB images in JPG format of 72 persons in slightly different positions with respect to the camera for a total of 20 expressions. RGB images were transformed in gray images within Matlab. The same was done for the other images with colors. In Figure 6.2 the first subject is reported in all expressions.

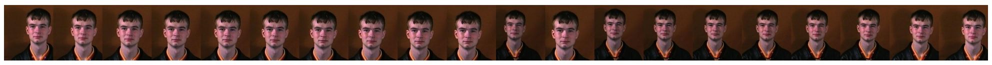

Fig. 6.2: Subject 1 of the Faces95 database photographed in 20 different expressions.

3. The **Faces96 database**, which is composed by 2261 images in JPG format of 119 persons at different distance with respect to the camera totalling 19 expressions. In Figure 6.3 the first subject is reported in all the expressions.

Fig. 6.3: Subject 1 of the Faces96 database.

4. The **COIL-20 database** composed by 20 objects, each of which is photographed in 72 different view angles. In Figure 6.4 object 1 in 15 different view angles is shown.

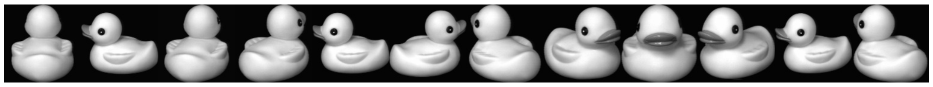

Fig. 6.4: Object 1 of the COIL-20 database.

5. The **Extended Yale database**, which contains 16380 grayscale images in PGM format of 28 persons. Each person is photographed in 9 poses and 65 illumination conditions for a total of 585 expressions. A shrunk version [4] of the Extended Yale database is used. In Figure 6.5 subject 1 is reported in 15 different expressions.

Fig. 6.5: Subject 1 of the Extended Yale database in 15 different expressions.

6. The **MIT-CBCL database**,which contains 3240 grayscale images of 10 persons in 324 different expressions.[5] The original pixel size of each image is 200×200. For our experiments the pixel size is reduced to 15×15. In Figure 6.6, 10 different expressions of person one are shown.

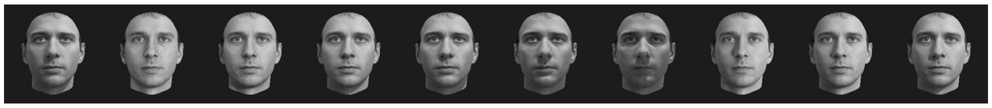

Fig. 6.6: Subject 1 of the MIT-CBCL database in 10 different expressions.

7. The **MNIST database**, from which we selected about 1901 grayscale 28×28 images of each of the 10 handwritten digits. In fact, for each digit we used 260 images as test set, and more images as training set, whose number depends on the used test-training splitting percentage. For this dataset the third variable cannot be given the interpretation of "expression". Indeed, there is no explicit correspondence between the same jth image of two different digits, so that images of each digit occur in a completely random order, though they all correspond to some sorts of stretching of a reference digit. We shall refer to this as *a lack of a classification feature*. In Figure 6.7, 10 different versions of the digit "three" are shown.

8. The **Fashion MNIST database**, which contains 70000 images of 10 different kinds of Zalando's articles. The Dataset is split in a training set, composed by 6000 images, and a test set, composed by 1000 images. For our experiments only a subset of the whole dataset is used. In particular we use all the images

[4]Each image has been reduced to the pixel size 20×15, instead of the original 640×480 size.
[5]In our experiments only the training set of the database is considered for generating the training and test phases.

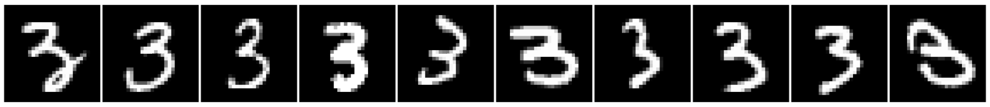

Fig. 6.7: Digit 3 of the MNIST database.

Fig. 6.8: First and second object of the Fashion MNIST database in the same "expression".

of the test set but only a subset of images of the training set, as required by the chosen test-training splitting percentage. In Figure 6.9 the first object is represented in 10 different conditions. Notice that, as for the MNIST, the variable expression is not defined, thus this variable lacks of a classification feature. For example, as shown in Figure 6.8, it is not possible to define the variable "expression 1", since expression 1 for the first object is different from expression 1 for the second object. Thus, with such data, we may expect HOSVD, TT3D and TT4D to achieve low classification success rates.

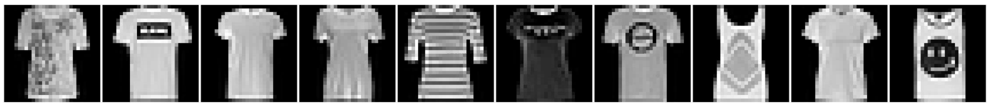

Fig. 6.9: First object of the Fashion MNIST database.

The characteristics of all databases are summarized in Table 6.1.

7. Numerical experiments. The validation of a classification algorithm needs to be addressed by using different perspectives. The most obvious way to evaluate the algorithm performance is to inspect its median success rate across several trials, as we are going to show in Table 7.1. However, other important measures should be taken into account to perform a fair comparison among different classification strategies. For example, CPU time, memory requirements and statistical classification parameters may be crucial in ranking the given methods, depending on the application where the algorithm is supposed to be used. For these reasons, in the next two subsections we report on our computational experience using the mentioned performance measures.

All numerical experiments were performed using Matlab [**?**].

7.1. A first Classification test. In this section we first report on our numerical experiments where we use the success rate as performance measure. To this end, we split each database in a training set and a test set. The split is made by taking the $s\%$ of the n_e expressions for each person as training set and the remaining ones as test

Database	pixel size	pixel size (vector format)	n_e	n_p
Orl	92×112	10304	10	40
COIL-20	128×128	16384	72	20
Faces95	180×200	36000	20	72
Faces96	196×196	38416	19	119
Ext'd Yale shrunk	20×15	300	585	28
MNIST	28×28	784	1901	10
Fashion MNIST	28×28	784	7000	10
MIT-CBCL	15×15	225	324	10

Table 6.1: Pixel size, number of expressions and persons of all databases.

set. The expressions used as training set are chosen randomly. We report results for $s = 80\%, 50\%$; except for HOSVD2, the tensor methods were not overly sensitive to the choice of this parameter. In Table 7.1 we report the success rate of all considered algorithms, namely matrix SVD, tensor SVD (HOSVD, HOSVD2), three- and four-dimensional TT (TT 3D and TT 4D) and simple least squares (LS) method. For all datasets except the large MNIST ones, all percentages correspond to the average of 20 consecutive runs.

$s\%$	Database	SVD	HOSVD	TT 3D	TT 4D	LS	HOSVD2
80	Orl	97.42%	94.44%	96.65%	96.69%	95.81%	**97.63**%
	COIL-20	**99.95**%	97.50%	99.35%	99.35%	99.17%	99.95%
	Faces95	90.00%	86.56%	87.15%	**91.03**%	86.44%	88.87%
	Faces96	100%	97.68%	**100**%	98.50%	97.23%	99.36%
	Ext'd Yale shrunk	59.59%	99.08%	99.24%	98.72%	**99.78**%	69.04%
	MNIST	62.55%	92.27%	91.05%	91.03%	**92.69**%	87.95%
	Fashion MNIST	49.19%	78.63%	79.20%	77.74%	**84.76**%	45.91%
	MIT-CBCL	23.52%	100%	**100**%	100%	100%	39.77%
50	Orl	94.84%	91.38%	92.96%	92.75%	89.93%	**95.20**%
	COIL-20	**99.84**%	96.10%	98.90%	96.71%	98.04%	99.24%
	Faces95	96.25%	**96.25**%	83.90%	87.50%	80.24%	88.87%
	Faces96	98.66%	97.34%	97.94%	**99.33**%	96.87%	98.04%
	Ext'd Yale shrunk	99.87%	98.70%	98.74%	98.99%	99.64%	**99.91**%
	MNIST	71.25%	**90.87**%	89.12%	89.12%	89.83%	37.68%
	Fashion MNIST	40.34%	74.58%	75.22%	74.57%	**81.31**%	33.16%
	MIT-CBCL	100%	100%	**100**%	100%	100%	99.98%

Table 7.1: Success rate (as percentage) of all classification algorithms.

Table 7.1 shows that SVD works well only when $n_i > n_e$, i.e. with the Orl, COIL-20, Faces95 and Faces96 datasets. However, in many applications it happens that $n_i < n_e$ because of the huge number of features available. This suggests that alternative strategies should be considered whenever the number of, say, expressions n_e is significantly larger than the number of pixels n_i or that the dataset is so heterogeneous that few columns are not enough to describe the whole dataset. As we can see from Table 7.1, some tensor methods (i.e. HOSVD, TT3D and TT4D) provide viable strategies. It is also interesting to observe that LS works quite well for the datasets where the third variable does not provide a real classification feature, that is for the two MNIST sets. HOSVD2 works well when $n_i < n_e$ if r and s are set equal to n_i and n_e respectively (i.e. no truncation is performed on $\mathcal{S}$) or as long as B^p is a tall matrix.

Memory requirements of the tensor methods can be quite different. Given the tensor $\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}$, the following table summarizes the minimum storage requirements:

Method	Memory allocations
HOSVD	$n_i n_e n_p + n_i^2 + n_e^2 + n_p^2$
TT 3D	$n_i^2 + n_i n_e n_p + n_p^2$

In particular, HOSVD requires to store the core tensor $\mathcal{S} \in \mathbb{R}^{n_i \times n_e \times n_p}$ and three orthonormal matrices $U^{(1)} \in \mathbb{R}^{n_i \times n_i}$, $U^{(2)} \in \mathbb{R}^{n_e \times n_e}$ and $U^{(3)} \in \mathbb{R}^{n_p \times n_p}$. On the other hand, TT 3D needs to store $G_1 \in \mathbb{R}^{n_i \times n_i}$, $\mathcal{G}_2 \in \mathbb{R}^{n_i \times n_e \times n_p}$ and $G_3 \in \mathbb{R}^{n_p \times n_p}$, thus leading to lower memory requirements.

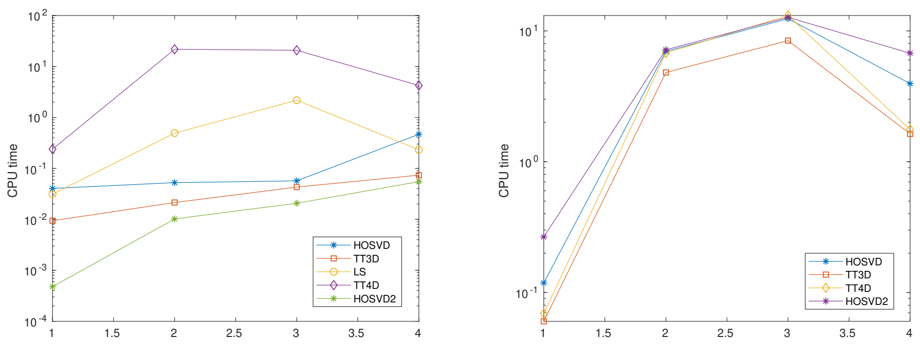

Fig. 7.1: C

PU time (secs) required by the tensor-based algorithms. Left: time to classify a person p in a specific expression e. Right: time to train the algorithms.

The two plots of Figure 7.1 report on the computational costs of the tensor-based methods on the four most time consuming datasets; for the very large dataset Fashion-MNIST runs were performed on a more powerful computer and therefore they are not reported. In Figure 7.1(left) the reported CPU time (in logarithmic scale) is the time required for the classification of a person p, in an expression e whenever appropriate. The plot shows that the CPU time of TT 3D and HOSVD2 is lower than for all other tensor-based methods. In Figure 7.1(right) the training time for all the tensor algorithms is reported. As we can observe the Tensor-Train algorithms require less CPU time than the other methods. Hence, this performance measure can help discriminate among methods whenever the classification success rate and memory requirements are comparable. Notice that the LS algorithm is not reported because there is no training time for this algorithm.

Remark 7.1. *To explore a different methodology, we have tested a simple Neural Network architecture (using the Matlab Deep Learning Toolbox [?]) with the following structure:*

- *One convolutional layer composed by 32* 3×3 *filters;*
- *A pooling layer (with a pooling region of width and height 2, and stride (2,2));*

- *A fully connected layer.*

On the MNIST database, this approach yielded a better classification performance (around 95%) than the tensor-based methods. As already mentioned, the lower performance of tensor-based methods on this dataset is expected, since one of the modes does not refer to a true feature. On the other hand, the CPU time required to train the Network and classify a single digit is around 35 seconds, almost 100 times the CPU time required for the new algorithm.

We also tested the same Network on the Orl database, obtaining significantly lower classification performance (77.38%) than our algorithm (96.65%) with higher CPU time (0.0043 sec for our algorithm, 18.9 sec for the Neural Network)

In spite of these negative results, we expect that more sophisticated Neural Network architectures may further improve performance, whereas computational costs may remain significantly higher than for the tensor-oriented approaches. □.

7.2. A Classification test in higher dimensional setting. In the previous experiments we have considered only two features per person, however in realistic applications a higher number of features may be available. In such situations the Tensor-Train format enables one to still store only third order tensors, whereas the HOSVD requires to allocate memory for a core tensor of the same order as $\mathcal{A}$.

In this section we analyze the classification performance of the Tensor-Train algorithm for a fifth-order tensor. To this end, we consider the Weizmann face image database composed by 28 subjects taken in 5 different view angles, 3 illumination conditions and 2 facial expressions. Each image size has been reduced from 352×512 to 14×20. In Figure 7.2 subject 1 in different conditions is represented.

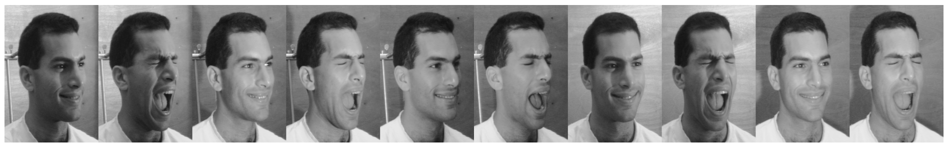

Fig. 7.2: Subject 1 of the Weizmann database.

The training set is stored as the tensor $\mathcal{A} \in \mathbb{R}^{n_i \times n_v \times n_{ill} \times n_e \times n_p}$. In a first classification test the testset is composed by n_p persons in n_e facial expressions, n_{ill} illuminations and 2 view angles. This means that 80% of the database is used for training and the remaining 20% is used as test set. In this setting 87.05% of the test images are correctly classified. Notice that TensorFaces with this dataset only achieves 80% success (for further details see [23]). In a second classification test the split between training and test sets is set to 90%/10% (i.e., just one view angle is used as test set). In this setting 91.07% of the examples are correctly classified.

7.3. Numerical experiments with truncated methods. In this section we report on our experiments with a truncated version of the 3D classification algorithms. We compare the truncated TT-SVD described in section 3 with the classification algorithms based on truncated SVD and truncated HOSVD2.

Figure 7.3 shows the percentage of success of the truncated procedure on eight databases, when different values of the truncation threshold π are used (see section 3). The displayed curves only report results for π such that $k \geq 1$. In HOSVD2 the truncation is applied to the columns of B^p, $p = 1, \cdots, n_p$ according to the truncation parameter π. Furthermore, following [18], $\mathcal{S}$ in (4.2) is replaced with $\hat{\mathcal{S}} = \mathcal{S}(1 : 48, 1 :$

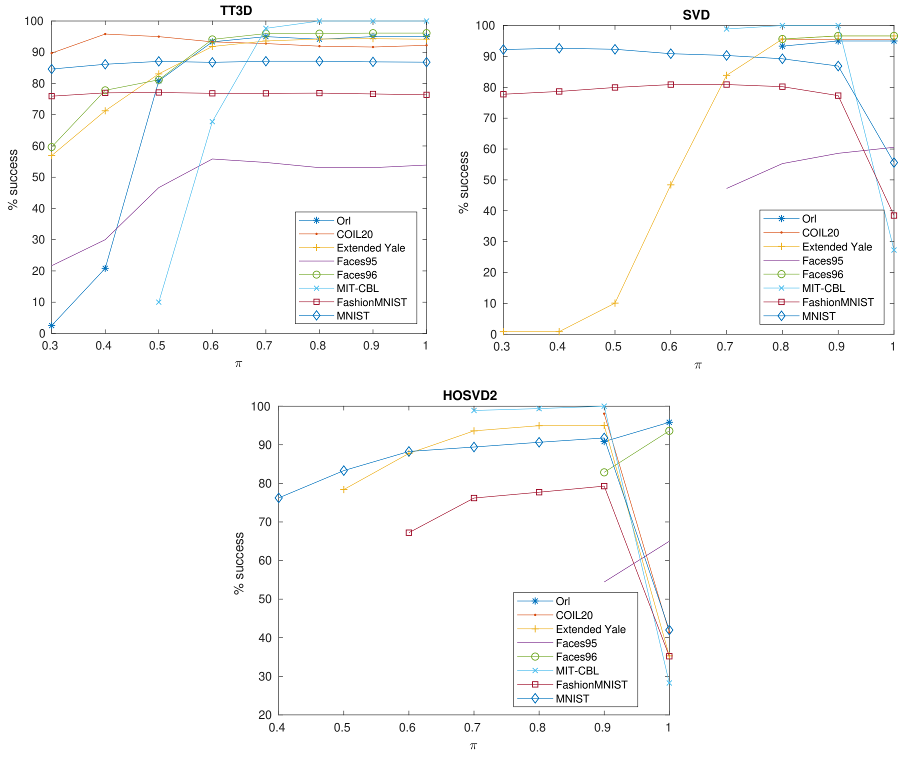

Fig. 7.3: Percentage of success of the truncated methods for different values of the truncation parameter π, tested on different databases.

64, :).

Several comments are in order. We observe that for some databases such as Orl, methods can behave quite differently for the same value of π. The decrease in the SVD performance for high values of π is related to the resulting dimensions of the singular vector matrix U_p: for $\pi \leq 0.9$ the matrix U_p is tall while for higher values of π it is square, leading to a projection failure. A similar reasoning holds for the performance degradation of HOSVD2: to express 90% of the variance (i.e. $\pi \geq 0.9$) of the MIT-CBL, COIL-20, MNIST and Fashion MNIST datasets we have to take all columns of B^p, making it square. Thus, for these algorithms there is an additional constraint on the truncation parameter, to ensure a good classification performance for several databases.

Figure 7.3 shows that the classification performance is not monotonic with respect to π. Nonetheless, for TT3D any $\pi \geq 0.6$ leads to a good classification performance.

The higher efficiency in terms of computational costs (CPU time), memory requirements together with the good recognition rate, favor the TT3D truncated version compared to the untruncated one. For instance, for $\pi = 0.9$ in the Extended Yale Database only 32% of the singular values are retained, yielding large memory savings.

7.4. Performance using statistical classification measures. Using an applied statistics terminology, the Face Recognition problem can be thought of as a multiclass classification problem, where the classes are the different persons of a specific database. For a database of n_p persons in n_e different expressions, we consider the splitting in a training set (n_p persons in $0.75n_e$ expressions) and a test set (n_p persons in $0.25n_e$ expressions). After the classification of all images in the test set has been completed, for each person p the following quantities can be computed, giving rise to the so-called “confusion matrix”:

- *true positive*(tp_p): number of subjects correctly classified as person p;
- *true negative* (tn_p): number of correctly recognized subjects that are not person p;
- *false positive* (fp_p): number of subjects incorrectly classified as person p;
- *false negative* (fn_p): number of subjects not recognized as person p.

Using these four quantities, the following classifier parameters can be computed:

$$Accuracy = \frac{1}{n_p}\sum_{p=1}^{n_p}\frac{tp_p + tn_p}{tp_p + tn_p + fp_p + fn_p}, \qquad Precision = \frac{1}{n_p}\sum_{p=1}^{n_p}\frac{tp_p}{tp_p + fp_p},$$

$$Recall = \frac{1}{n_p}\sum_{p=1}^{n_p}\frac{tp_p}{tp_p + fn_p},$$

These parameters provide a measure of reliability of the employed algorithm: the closer the parameter is to 100% the more robust the corresponding classification strategy is. Depending on the realistic application for which we want to use the classification algorithm, one can be interested in maximizing one of these three quantities. For example, in situations such as border controls or medical tests, one may be interested in maximizing the recall, since this means minimizing the number of images of person p that are not recognized as person p.

In Table 7.2 *precision*, *accuracy* and *recall* are displayed, using macro-averaging (for further details see [20]). The results show an overall different ranking of the methods when these measures are considered, with respect to the percentages in Table 7.1. HOSVD appears to be the most reliable strategy for all three parameters for the MNIST database, whereas for all other datasets tensor-train based algorithms have better performance. It is also interesting that, although the three parameters measure truly different things, the relative ranking of all algorithms does not change much going from, say, Accuracy to Recall.

HOSVD2 was not included in the tests in Table 7.2, because its classification performance strongly depends on the truncation parameter.

		Orl	COIL	Faces95	Faces96	Ext’d	MNIST	Fashion
measure	method		-20			Yale		MNIST
Accuracy	HOSVD	99.71%	**99.77%**	99.66%	99.97%	99.93%	**98.18%**	95.26%
	TT 3D	**99.78%**	**99.77%**	**99.71%**	**99.98%**	**99.94%**	97.83%	**95.39%**
	TT 4D	99.77%	**99.77%**	99.60%	**99.98%**	99.94%	97.67%	**95.39%**
Precision	HOSVD	76.62%	**98.03%**	91.10%	98.60%	99.06%	**91.18%**	78.30%
	TT 3D	**96.73%**	**98.03%**	**92.53%**	99.07%	99.17%	89.89%	**78.53%**
	TT 4D	96.58%	**98.03%**	90.01%	**99.20%**	**99.20%**	89.00%	**78.53%**
Recall	HOSVD	94.29%	97.67%	87.67%	98.29%	99.01%	**90.90%**	76.31%
	TT 3D	**95.62%**	**97.69%**	**89.60%**	98.85%	99.12%	89.10%	**76.93%**
	TT 4D	95.50%	**97.69%**	85.62%	**99.01%**	**99.16%**	88.32%	**76.93%**

Table 7.2: Accuracy, precision and recall for the HOSVD, TT3D and TT4D classification algorithms.

8. Conclusions and perspectives.

Based on the Tensor-Train decomposition of the given dataset, we have proposed a new algebraic strategy for face and object recognition, both in a three-dimensional and four-dimensional setting. The use of the TT form allows one to easily treat truncation, which reduces both CPU time and memory requirements, without sacrificing the recognition success rate. Moreover, the TT-based algorithm is preferable to those based on HOSVD since it does not suffer from the curse of dimensionality; in particular, it naturally extends to more than three dimensions, thus allowing for the inclusion of additional features as extra dimensions, as we did for the Weizmann database. Indeed, for face recognition other features could be considered, such as different age or backdrop image sets.

Our computational experiments on nine different datasets seem to show that using the Tensor-Train form allows one to achieve good classification success for comparable memory requirements (in the full case) and smaller CPU time with respect to the now classical tensor based HOSVD.

Due to the relatively low computational costs of the TT methodology, we also envision using the proposed classification algorithm as a preliminary first pass to a significantly more expensive but even more accurate procedure, e.g., within deep learning methodology, to narrow the classification dataset.

9. Acknowledgements.

Both authors are members of the GNCS-Indam activity group. Its financial support is fully acknowledged. We thank one anonymous reviewer for several insightful remarks and important pointers to the literature.

REFERENCES

[1] D. P. Mandic A. Cichocki, A. H. Phan, C. F. Caiafa, G. Zhou, Q. Zhao, and De Lathauwer L. Tensor decompositions for signal processing applications from two-way to multiway component analysis. *CoRR*, abs/1403.4462, 2014.

[2] A. Osokin A. Novikov, D. Podoprikhin and D. P. Vetrov. Tensorizing neural networks. *CoRR*, abs/1509.06569, 2015.

[3] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 3.0-dev. Available online, October 2017.

[4] M. Boussé, N. Vervliet, O. Debals, and L. De Lathauwer. Face recognition as a Kronecker product equation. In *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 1–5, Dec 2017.

[5] E. Ceulemans and H. Kiers. Selecting among three-mode principal component models of different types and complexities: A numerical convex hull based method. *The British Journal of mathematical and statistical psychology*, 59(1):133–50, 06 2006.

[6] A. Cichocki. Era of big data processing: A new approach via tensor networks and tensor decompositions. *CoRR*, abs/1403.2048, 2014.

[7] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[8] K. Delac, M. Grgic, and S. Grgic. *Recent advances in Face Recognition.* BoD–Books on Demand, 2008.

[9] L. Eldén. *Matrix Methods in Data Mining and Pattern Recognition (Fundamentals of Algorithms).* Society for Industrial and Applied Mathematics, USA, 2007.

[10] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013.

[11] W. Gong, M. Sapienza, and F. Cuzzolin. Fisher Tensor Decomposition for Unconstrained Gait Recognition. In *ECML/PKDD Workshop*, 01 2013.

[12] N. Hao, M. E. Kilmer, K. Braman, and R. C. Hoover. Facial recognition using tensor-tensor decompositions. *SIAM Journal on Imaging Sciences*, 6(1):437–463, 2013.

[13] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[14] S. Krämer. A Geometric Description of Feasible Singular Values in the Tensor Train Format. 2019.

[15] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*,

33(5):2295–2317, 2011.

[16] I. V. Oseledets and E. Tyrtyshnikov. Tt-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010.

[17] A. H. Phan, A. Cichocki, A. Uschmajew, P. Tichavský, G. Luta, and D. P. Mandic. Tensor networks for latent variable analysis. part I: algorithms for tensor train decomposition. *CoRR*, abs/1609.09230, 2016.

[18] B. Savas and L. Eldén. Handwritten digit classification using higher order singular value decomposition. *Pattern Recognition*, 40(3):993 – 1003, 2007.

[19] A. W. Senior and R. M. Bolle. *Face Recognition and its Application*, chapter 10, pages 83–97. Springer USA, Boston, MA, 2002.

[20] M. Sokolova and G. Lapalme. A Systematic Analysis of Performance Measures for Classification Tasks. *Inf. Process. Manage.*, 45(4):427–437, July 2009.

[21] D. Tock. Tensor decomposition and its applications. Master's thesis, University of Chester, 2010.

[22] M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.

[23] M. A. O. Vasilescu and D. Terzopoulos. Multilinear image analysis for facial recognition. In *Object recognition supported by user interaction for service robots*, volume 2, pages 511–514. IEEE, 2002.

[24] D. Krompass Y. Yang and V. Tresp. Tensor-train recurrent neural networks for video classification. *CoRR*, abs/1707.01786, 2017.