Face Recognition using HOSVD

Oumayma El Hit, Rasmus Hogslätt

December 8, 2022

1 Introduction

Face recognition is nowadays largely used in several fields[Cyb] such as access control, security and surveillance, health and safety, time and attendance, smart retail and personalized customer experiences as well as law enforcement. Having well developed automatic procedures for face recognition is crucial, as it is used in critical fields such as crime detection which require exactitude and efficiency. Face recognition can help in significantly improving secure and accurate predictions based on the AI methods, even when the target persons are wearing face coverings or face masks. These are things which humans are not always qualified to do. This project consists of implementing a classification algorithm of images, based on a given data set using higher order singlar value decomposition ,*HOSVD*.

1.1 Problem

Given an image of a person, determine whether that person is in a database and if so, determine which person in the database it is, by using *HOSVD*.

1.2 Data set

The data set used is a collection of images of size 112x78 pixels, described as a vector of size $n_i = 8736$ pixels. Images portray $n_p = 10$ different persons, taken in $n_e = 11$ different expressions. The data was provided as a *Matlab* multidimensional array by[Eld19] and converted into a three mode tensor. The tensor modes are referred to as the *image* mode, the *expression* mode and *person* mode respectively. The data was originally extracted from the Yale Face Database[Yal], from which, for the sake of simplicity, less compute and lower use of memory, only a subset of persons were included. An example of such an image is shown in Figure 1.

Figure 1: Person 10, in expression 10 in Yale's database.

2 Theoretical foundation

2.1 Tensors

Applications requiring accumulation of data often organize it into meaningful categories. When there are one or two categories, vectors and matrices provide enough dimensions to sensible structure the data. However, many applications can't sensible fit all categories into only two dimensions, thus tensors

are used. Each dimensions of a tensor is called a mode, thus a two mode tensor is equivalent to a matrix, whereas a one mode tensor is a vector. A tensor can have any number of modes but for this report, only one to three mode tensors are considered. The ideas however, can be extended into higher mode tensors.

2.2 Tensor operations

2.2.1 Inner product and norm

The inner product of two tensors $\mathcal{A}$ and $\mathcal{B}$ is defined by

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i,j,k} a_{ijk} b_{ijk} \tag{1}$$

and the norm norm of a tensor $\mathcal{A}$ is given by

$$||\mathcal{A}||_F = \langle \mathcal{A}, \mathcal{A} \rangle^{1/2} \tag{2}$$

2.2.2 Tensor mode multiplication

Tensors introduce the concept of n-mode multiplication, which is denoted as $\mathcal{A} \times_n U$ where $\mathcal{A}$ is a tensor and U is a matrix. For a three mode tensor $\mathcal{A} \in R^{l \times m \times n}$ multiplied by a matrix $U \in R^{m_o \times m}$ in the second mode,

$$(\mathcal{A} \times_2 U)(i_1, j, i_3) = \sum_{k=1}^{m} u_{j,k} a_{i_1,k,i_3} \tag{3}$$

is equivalent to multiplying all row vectors of $\mathcal{A}$ by matrix U. If the rows of the tensor were to be seen as matrices, the second mode multiplication would be the same as multiplying from the right,

$$A_{row} \times_2 U = A_{row} U^T \tag{4}$$

and a one mode multiplication would be analogous to a left side matrix multiplication. The third mode multiplication would thus be a similar to a matrix multiplication from its dimension.
Similarly to the tensor mode multiplication by a matrix as described in equation 3, tensor mode multiplication is also defined for vectors. Given a vector $v \in R^n$, the tensor mode multiplication in mode one, $\mathcal{A} \times_1 v$, is given by

$$(\mathcal{A} \times_1 v)(j, i_2, i_3) = \sum_{k=1}^{n} v_k a_{k,i_2,i_3} \tag{5}$$

which corresponds to multiplying vector v by each fiber along mode one in $\mathcal{A}$.

2.3 Tensor unfolding

Many mathematical operations have been developed to achieve various tasks, and many of those are limited to one or perhaps two dimensions. Thus, it's often useful to restructure a tensor into a matrix, which is referred to as unfolding a tensor. Unfolding with respect to a given mode, where chosen mode becomes the first dimension of the matrix, and entries of data in other modes are handled cyclically. Therefore, unfolding along modes one to three yields matrices of dimensions

$$\begin{aligned} U_1 &\in R^{l \times mn} \\ U_2 &\in R^{m \times ln} \\ U_3 &\in R^{n \times lm} \end{aligned} \tag{6}$$

where U_1 would contain slices $\mathcal{A}(:, i, :)$ arranged into one long matrix.

2.4 HOSVD

A three mode tensor $\mathcal{A} \in R^{l \times m \times n}$ can be decomposed using higher order singular value decomposition or HOSVD into a core tensor of equal size and three square orthogonal matrices according to equation 7

$$\mathcal{A} = \mathcal{S} \times_1 U_1 \times_2 U_2 \times_3 U_3 \tag{7}$$

where $U_1 \in R^{l \times l}$, $U_2 \in R^{m \times m}$, $U_3 \in R^{n \times n}$ and $\mathcal{S} \in R^{l \times m \times n}$. The singular values of $\mathcal{S}$ in mode one is given by

$$\sigma_i = ||\mathcal{S}(i, :, :)||_F, i = 1...l \tag{8}$$

and ordered by size. The singular values for the other modes given similarly. This property means that singular values containted within the top corner of $\mathcal{S}$ are more significant than those in the other parts. This introduces the concept of truncated HOSVD's, where only up to a given rank k is kept from the tensor.
$\mathcal{S}$ is also all-orthogonal, meaning that the dot product between any two slices in the same mode are orthogonal, i.e.

$$\langle \mathcal{S}(i, :, :), \mathcal{S}(j, :, :) \rangle = \langle \mathcal{S}(:, i, :), \mathcal{S}(:, j, :) \rangle = \langle \mathcal{S}(:, :, i), \mathcal{S}(:, :, j) \rangle = 0 \tag{9}$$

where $i \neq j$.

2.5 QR decomposition

A matrix $U \in R^{m \times n}$ can be decomposed according to equation 10

$$U = QR \tag{10}$$

where columns of Q are orthogonal unit vectors and R is an upper triangular matrix. If $m > n$, only n first columns of Q and n first rows of R need to be kept, as remaining data does not contain any meaningful values. These matrices can further be used to solve least squares problems as in equation 11.

$$min_x = ||Ax - b||_2 \tag{11}$$

3 Implementation

To determine whether an image of a person was in a database, a program was made using *Matlab* with an add on called *Tensor Toolbox*[Cor], which provided common utility functions for tensors.

3.1 Overview

The program was divided into three main components. A training phase, a classifying algorithm and a testing phase that relied on the previous components.

3.1.1 Training phase

The purpose of the training phase was to prepare data for the classifier in the next stage. Data from the Yale[**?**] data base was loaded into a three mode tensor in *Matlab*. It was then divided into two a training part containing data belonging to the first eight expressions, and the testing data containing the remaining three.
The training part was unfolded along the three modes, and *Matlab's* built in thin singular value decomposition algorithm was applied to the unfolded matrices, which resulted in the orthogonal matrices $U_1 \in R^{n_i \times n_e n_p}, U_2 \in R^{n_e \times n_e}$ and $U_3 \in R^{n_p \times n_p}$. These are the same as in equation 7, which were used to construct a tensor core S according to equation 12

$$S = A \times_1 U_1^T \times_2 U_2^T \times_3 U_3^T \tag{12}$$

by using tensor mode multiplication provided by the *Tensor Toolbox*.
Multiplying the tensor core S in the second mode as in equation 13

$$B = S \times_2 U_2 \tag{13}$$

gives a new tensor $B \in R^{n_e n_p} \times n_e \times n_p$ where each slice along the expression mode is a matrix of vectors providing a base for the indexed expression. Due to the singular values being ordered by the singular value decomposition, these vectors are ordered by significance. This property was further used to construct a truncated version of the B tensor called $B_k \in R^{k \times n_e \times n_p}$ given by

$$B_k = B(1:k,:,:) \tag{14}$$

where k was the number of vectors included in the truncation.
Matlab's thin QR decomposition was then applied to each expression slice of B and B_k, resulting in matrices Q, R and Q_k and R_k respectively.
The U_1 matrix was also truncated, giving a matrix $U_{1k} \in R^{n_i \times k}$. Note that the reason both the non truncated and truncated version were kept was for testing performance using different truncation options.

3.1.2 Classifying algorithm

The classifying algorithm was implemented as a function in *Matlab*, of the form

$$person = function(testImage, Q, R, F, tolerance, H) \tag{15}$$

where *testImage* was an image given in the format of a vector of size n_i, Q, R, F and H were the matrices computed in the training phase and *tolerance* was a threshold for when the algorithm should classify a test image as a person in the training data or not.
The test image was multiplied by transposed F

$$\hat{testImage} = F^T * testImage \tag{16}$$

which resulted in a set of coordinates describing the content of the image. Similarly, each row vector of H held a set of coordinates for persons in the training data. The function then iterated through each expression of the QR decompositions and solved the least squares problem

$$R_e * solution = Q_e^T * \hat{testImage} \tag{17}$$

where e denotes the index begin processed, using *Matlab's* built in solver. After finding the solution, the norm

$$||solution - H_p|| \tag{18}$$

was calculated for each person p, where H_p were coordinates for actual persons in the data. If the norm was lower than the tolerance, the algorithm would the the person of the test image as person p from the data. If it was not lower than the tolerance, it would keep iterating over the next expression. If no solution was lowered than given tolerance, the image was considered not to contain a person in the data.

3.1.3 Testing phase

The testing phase would let a user choose an image of person from the data by entering two numbers indicating what person and what expression to choose. Then, that image was passed into the classifying function along with the training parameters computed in the previous stage. For comparison, the image was classified using both non truncated and truncated training parameters. The user could also adjust tolerances and retrain the algorithm with a different value k for the truncation.

3.1.4 Pseudo code

The pseudo code of the program can be written as follows

Preprocessing step : consists of computing and save the thin QR decompositions of all the B_e matrices, $B_e = Q_e R_e$, $e = 1, 2, 3, ...n_e$.
$\%z$ is a test image.
compute $\hat{z} = F^T z$.
for $e = 1, 2, ..., n_e$

Solve $R_e \alpha_e = {Q_e}^T \hat{z}$ for α_e.
for $p = 1, 2, ..., n_p$
If $\|\alpha_e - h_p\|_2 < tol$, then classify as person p and **stop**.
end
end.

4 Results

Results after running the testing phase with various parameters are shown in tables 1, 2 and 3.

Expression 10	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Performance [%]
Tolerance = 0.5	1	2	3	4	5	6	7	-	9	10	90
Tolerance = 0.25	-	-	-	4	5	-	-	-	-	-	20
Expression 9											
Tolerance = 0.25	1	2	3	4	5	6	7	8	-	10	90
Expression 11											
Tolerance = 0.25	-	-	-	-	5	-	-	-	9	-	20

Table 1: Performance without truncation.

Expression 10	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Performance [%]
Tolerance = 0.5	1	2	3	4	5	6	-	8	9	10	90
Tolerance = 0.25	-	-	-	4	5	-	7	-	-	-	30
Expression 9											
Tolerance = 0.25	1	2	3	4	5	6	7	-	-	10	80
Expression 11											
Tolerance = 0.25	1	-	-	-	5	6	-	8	-	10	50

Table 2: Performance with truncation k = 15.

Expression 10	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Performance [%]
$k = 5$	-	-	-	-	-	-	-	-	9	-	10
$k = 10$	1	-	-	-	5	6	7	-	-	-	40
$k = 15$	-	-	-	4	5	-	7	-	-	-	30

Table 3: Performance with tolerance 0.5 and truncation by number k.

5 Discussion and conclusion

As seen in tables 1, 2 and 3, the accuracy of the algorithm varies. However, some notable points are that the accuracy tends to increase when k is increased. This is likely due to the fact that allowing a larger vector subspace allows better representation of the various persons. Thus, the tolerance threshold is more likely to be achievable when comparing the test image's coordinates to the coordinates of the persons. It can also be seen that an increased tolerance yields higher performance, however, it is likely that an increase in tolerance may cause the algorithm to predict the wrong person, i.e. that person 5 may be classified as person 7 for example. This was, however, not seen in the conducted experiments. The results only ever predicted a person to be the correct person, or not exist at all, and would never predict a person to be a different person. This was likely due only using a small subset of Yale's full database, thus, there were fewer wrong persons possible to predict. Therefore, the performance may be artificially high.

The current implementation does not care what expression is being predicted, only that the person is found within a tolerance. It would have been possible to go through each expression and make sure to find the best fit solution, and thus increase the probability that the person was predicted in the most similar expression. Making use of a best fit for all expressions, while also checking for a tolerance, would have been interesting to investigate further.

The project showed that tensors are useful structures when it comes to representing data, that doesn't easily fit into fewer than three dimensions. In this case, three mode tensors were utilized, but the concepts used could easily have been used to model data in even higher dimensions. For example, companies such as Google most likely have tensor data in magnitudes more modes than three. *HOSVD* for tensor approximation also proved to be an effective method for reducing size of these large data types, while also keeping many of the relevant features in the data.

It was also seen that the non truncated predictions were more accurate the those that were truncated. However, a truncated version may still be more preferable depending on the size of the data. The truncated version became significantly much smaller than the non truncated version, which lowered both computational and memory cost. For this application, the size of the data was relatively small, but for larger applications, this would likely make a huge impact on performance relative to required compute power and memory storage.

References

[Cor] Sandia Corporation. Tensor toolbox for matlab, version 3.4.

[Cyb] Cyberlink. Top 7 use cases for facial recognition in 2023.

[Eld19] Lars Eldén. Matrix methods in data mining and pattern recognition. *SIAM*, 2nd edition, 2019.

[Yal] Yale. The yale face database.