Universität Trier | Fakultät IV | Ms. Data Science

Exploring Videos as Tensors

Jean C. Fernández, Data Science, Matricul-Nr. 1510459,
Javier A. Jáquez, Data Science, Matricul-Nr. 1510435
Abdelrahman Elmorsy, Data Science, Matricul-Nr. 1510423

Trier, 03/ 2021

Abstract

Our goal as Data Scientist is to explore video data structures using the relation between tensors and their core tensors of the Tucker decomposition as a way to reduce their dimension. For this, an experiment using two core Tucker cores from a video data-set are used in a clustering algorithm, where the assigned clusters are compared with respect to their *true-labels* (Some previously known information about the data-set). Using the sample Tucker cores, the required computation time to run the clustering algorithm became something trivial. Without using any feature engineering the clusters and labels shared a satisfying agreement corrected for randomness of cluster choice. Implying that the Tucker cores are indeed a lower-rank representation of videos.

Contents

1 Introduction

Nowadays, data generation increases by the second in larger volumes. With the wide spread of cameras, CCTV and smartphone, most of this content ends up online on places like social media, giving us easy access to it. Processing such data and tapping into the underlying useful information and patterns is essential for many business models and developing technologies like: self-driving cars, drones, satellites, decision-making processes, etc. However, videos are hard to process making the analysis cumbersome and expensive due to their size, noise or redundant information.

Videos are high dimensional data, meaning that they have many features compared to the number of observations available for analysis. This might impair the analysis or model learning process. Fortunately, tensor decompositions offer a particularly less expensive way to represent videos. Through decomposition, we obtain core tensors which are significantly lower in dimensions than then original tensors. Moreover, they offer a mathematically efficient way of compressing the volume of data thus reducing the overall analysis cost, while keeping most of the underlying information and variation of the original video intact.

In this paper, we try to cover the theoretical background of: Tensors and their structure, tensor decomposition techniques (Tucker and CP decomposition), and finally representing videos as tensors. This with the purpose of testing the usefulness of core tensors as a representation of videos. To this an experiment was carried out using a data-set with videos which are similar. We decomposed this videos with the explained tensor framework and clustered them for comparison. At the end our results supported this claim.

2 Tensor

A tensor is a multidimensional array. In other words, an N^{th}-order tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ describes the relation between N objects, which may be vectors, scalars or other tensors. According to Bowen and Wang [1], if we have a collection of vector spaces $v_1, \cdots, v_n$, then we can say that A is an n-linear function such that

$$A : v_1, \cdots, v_n \rightarrow \mathbb{R}, \tag{1}$$

which is linear for each of its variables while others are constant, but if vector spaces $V_1, \cdots, V_n$ equal the vector space V or its dual vector space V^*, then A is said to be a tensor on V or a tensor of order (p, q) on V if $p, q \in \mathbb{Z}^+$ such that

$$A : V \times \cdots \times V \times V^* \times \cdots \times V^* \rightarrow \mathbb{R} \text{ is a } (p+q) \text{ linear function.} \tag{2}$$

In this paper, we follow the notation used by Kolda and Bader [2]. We denote vectors by lower case letter such as x, matrices by upper case letters such as X and a higher order tensor by Euler script letters $\mathscr{X}$.

2.1 Tensor order

The order of tensor or mode, as stated by Kolda and Bader [2], is a term used to refer to the number of dimensions represented in the tensor. A tensor of the 0th order is a simply a scalar, a 1st order tensor is a vector, a 2nd order vector is a matrix and 3rd, higher or N^{th}-order tensor is called higher-order tensor. Moreover, an Nth-order tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ can be represented by the vector outer product '$\circ$' or the product of its corresponding N vectors, i.e.,

$$\mathscr{X} = a^{(1)} \circ a^{(2)} \circ \cdots \circ a^{(N)}. \tag{3}$$

The vector outer product $\circ$ means that every tensor's element equals the product of its corresponding vector elements. Generally, tensors are used to represent information in N number of dimensions. A vector x carries information about one dimension or in one direction, a matrix x carries information about two dimensions or in two directions,a 3rd order tensor $\mathscr{X}$ carries information about three dimensions or in three directions and so on. As the order increases, it becomes harder to visualize the representation, Fig. 1 [3] may provide better visualization.

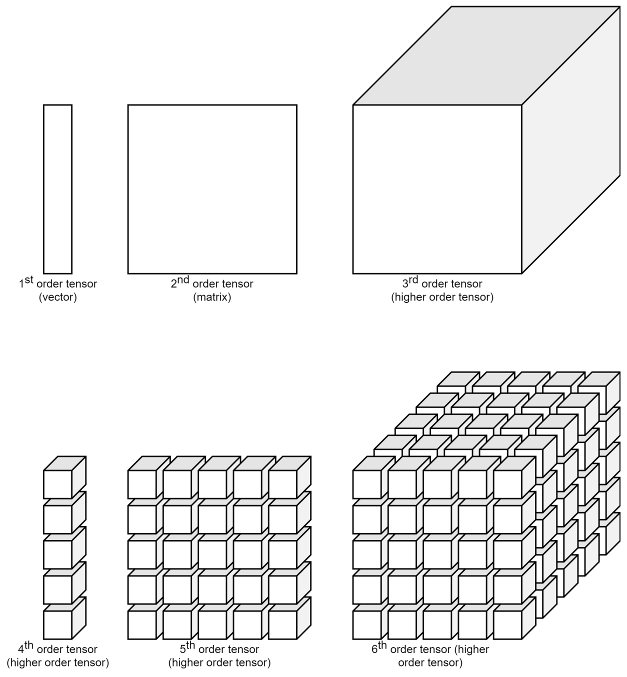
1st order tensor (vector)
2nd order tensor (matrix)
3rd order tensor (higher order tensor)
4th order tensor (higher order tensor)
5th order tensor (higher order tensor)
6th order tensor (higher order tensor)

Fig. 1: Tensors with varying orders (dimensions)

2.2 Tensors subarrays

In matrices, a subarray is formed when one of the indices is fixed and it referrers to rows and columns and they are denoted by using a colon for all elements of the other dimensions, e.g., for a matrix X, the i$^{\text{th}}$ row is denoted by $a_{i:}$ and the j$^{\text{th}}$ column as $a_{:j}$. Fibers are the equivalent in higher-order tensors, which are denoted by fixing all of the indices but one, in a 3$^{\text{rd}}$ order tensor $\mathscr{X}$, we get rows $x_{i:k}$, columns $x_{:jk}$ and tube fibers $x_{ij:}$. A slice is a 2-dimensional section of a tensor, which is the result of fixing all but two indices. There are horizontal, lateral and frontal slices, in a 3$^{\text{rd}}$ order tensor $\mathscr{X}$, this will represent a matrix and is denoted by $X_{i::}$, $X_{:j:}$ and $X_{::k}$ respectively. Fibers and slices for a 3$^{\text{rd}}$-order tensor are shown in the Fig. 2 [2].

2.3 Tensor Multiplication

Now, as Bader and Kolda [4] admit, Higher order tensor multiplication is much more complex than that for vectors or matrices, the issue being how to determine the dimensions of the multiplied tensor and the dimensions and order of the resulting tensor.

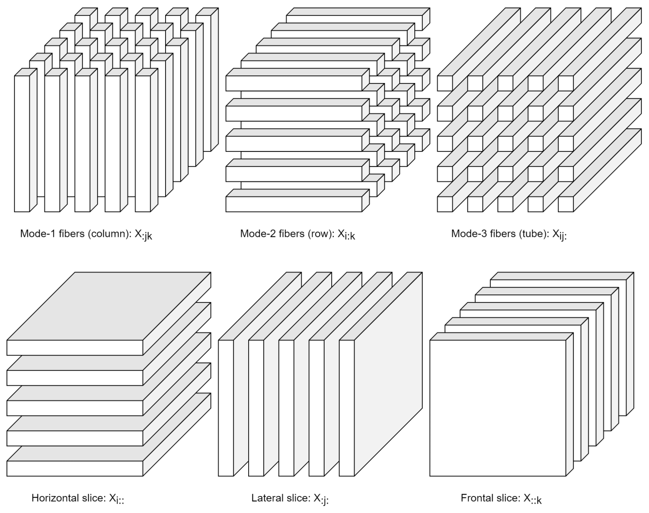
Mode-1 fibers (column): $X_{:jk}$
Mode-2 fibers (row): $X_{i:k}$
Mode-3 fibers (tube): $X_{ij:}$
Horizontal slice: $X_{i::}$
Lateral slice: $X_{:j:}$
Frontal slice: $X_{::k}$

Fig. 2: Fibers and Slices

2.3.1 The n-mode Product

The n-mode product refers to multiplying a higher-order tensor by a lower-order tensor (vector or matrix) in mode n. The n-mode product of an N$^{\text{th}}$ tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a vector $u \in \mathbb{R}^{I_n \times 1}$ results in a tensor $\mathscr{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times 1 \times I_{n+1} \times \cdots \times I_N}$ and is denoted by $\mathscr{B} = \mathscr{A} \times_n u^T$, where each entry in $\mathscr{B}$ is a sum of the resulting product of corresponding elements in $\mathscr{A}$ and u as follows:

$$\mathscr{B}(i_1, \cdots, i_{n-1}, 1, i_{n+1}, \cdots, i_N) = \sum_{i=}^{n} \mathscr{A}(i_i, \cdots, i_N) \cdot u(i_n) \tag{4}$$

Mode-n product of an N$^{\text{th}}$-order tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $U \in \mathbb{R}^{J_n \times I_n}$ results in a tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_n \times I_{n+1} \times \cdots \times I_N}$and is denoted by $\mathscr{C} = \mathscr{A} \times_n U$, where each entry in $\mathscr{C}$ is a sum of the resulting product of corresponding elements in $\mathscr{A}$ and U as follows:

$$\mathscr{C}(i_1, \cdots, i_{n-1}, j_n, i_{n+1}, \cdots, i_N) = \sum_{i=1}^{n} \mathscr{A}(i_i, \cdots, i_N) \cdot U(j_n, i_n) \tag{5}$$

which is the same as multiplying each n-mode vector of $\mathscr{A}$ by U. De Lathauwer, De Moor and Vandewalle [5] show properties of tensor matrix multiplication as follows:
Property 1, for tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, matrix $U \in \mathbb{R}^{J_n \times I_n}$ and matrix $V \in \mathbb{R}^{J_m \times I_m}$, where $m \neq n$ is

$$(\mathscr{A} \times_m U) \times_n V = (\mathscr{A} \times_n V) \times_m U. \tag{6}$$

Property 2, for tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, matrix $U \in \mathbb{R}^{J_n \times I_n}$ and matrix $V \in \mathbb{R}^{K_n \times I_n}$ is

$$(\mathscr{A} \times_n U) \times_n V = \mathscr{A} \times_n (V \cdot U). \tag{7}$$

The product of a tensor and a vector is seen as a special case of tensor matrix multiplication were vector u is nothing but U with $J_n = 1$ and it is the inner product of mode-1 vectors of $\mathscr{A}$ and u.

2.3.2 Kronecker and Khatri–Rao Products

There are several other matrix product technique that are also relevant for tensor multiplication, which are introduced in the following. [6] [7]

2.3.3 Kronecker

Kronecker product for two matrices $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{K \times L}$ is denoted by $A \otimes B$ where the resulting matrix is of size $(IK) \times (JL)$ such that

$$A \otimes B = \begin{bmatrix} a_1 1B & a_1 2B & \cdots & a_1 JB \\ a_2 1B & a_2 2B & \cdots & a_2 JB \\ \vdots & \vdots & \ddots & \vdots \\ a_I 1B & a_I 2B & \cdots & a_I JjB \end{bmatrix} \tag{8}$$

$$= [a_1 \otimes b_1 \quad a_1 \otimes b_2 \quad a_1 \otimes b_3 \quad \cdots \quad a_J \otimes b_{L-1} \quad a_J \otimes b_L] \tag{9}$$

2.3.4 Khatri–Rao

Khatri–Rao is a Kronecker product for two matrices with equal dimensions for their columns $A \in \mathbb{R}^{I \times K}$ and $B \in \mathbb{R}^{J \times K}$ is denoted by $A \odot B$ where the resulting matrix is of size $(IJ) \times (K)$ such that

$$A \odot B = [a_1 \otimes b_1 \quad a_2 \otimes b_3 \quad \cdots \quad a_K \otimes b_K] \tag{10}$$

3 Tensor Decomposition

Decomposition are a way to express tensors as the sum of meaningful parts. Which can then be used for data analysis. We want to find a low rank model which is a good descriptor of the data.

3.1 CP Decomposition

In 1927 Frank Hitchcock, wrote a paper [8] where he proposed to express tensors as a sum of a finite number of rank one tensors. But it was not until 1970 that the concept became popular. In this year two groups reinvented the idea. One group was led by Caroll and Chang [9] (CANDECOMP), the other was led by Harshman [10] (PARAFAC). In order to address both of these influential groups the term CP Decomposition was created.

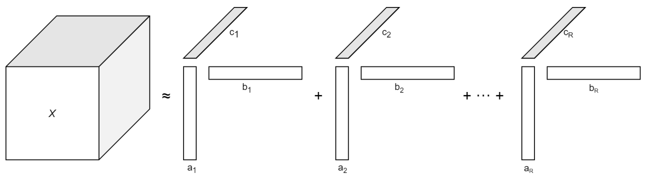

Fig. 3: CP decomposition in the 3-way case. Source: [2]

For ease of comprehension and explanation we will focus on the three-way case which we can see in Fig. 3. In CP decomposition a tensor is factorized into a sum of component rank one tensors. In the third order tensor we have $\mathscr{X} \in \mathbb{R}^{I \times J \times K}$

$$\mathscr{X} \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r. \tag{11}$$

Where R is a positive integer and $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$, and $c_r \in \mathbb{R}^K$ for $r = 1, ..., R$.

Equation (11) can be rewritten elementwise as

$$\mathscr{X}_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}. \tag{12}$$

for $i = 1, ..., I, j = 1, ..., J, k = 1, ..., K$. The idea in CP Decomposition is to look for factor Matrices A, B, C which best describe the data. For this we take the vectors from the rank one components and use them as columns of the factor matrices. In the following way: $A = [a_1, a_2, ...a_R]$. The same is then done for B and C.

In Kolda [2] the following notation is used to represent the CP Model

$$\mathscr{X} \approx [\![A, B, C]\!] \equiv \sum_{r=1}^{R} a_r \circ b_r \circ c_r. \tag{13}$$

Sometimes the columns of A, B, and C are normalized to length one with the weights absorbed into the vector$\lambda \in \mathbb{R}^R$, in the form:

$$\mathscr{X} \approx [\![\lambda; A, B, C]\!] \equiv \sum_{r=1}^{R} \lambda_r a_r \circ b_r \circ c_r. \tag{14}$$

This can also be written in matricized form, where $X_{(i)}$ represents one of the modes

$$\begin{aligned} X_{(1)} &\approx A(C \odot B)^T, \\ X_{(2)} &\approx B(C \odot A)^T, \\ X_{(3)} &\approx C(B \odot A)^T. \end{aligned} \tag{15}$$

3.1.1 Tensor Rank

The rank of $\mathscr{X}$ defines the smallest number of rank-1 tensors that can be used to generate $\mathscr{X}$ as the sum of all said rank-1 tensors, and is denoted by rank($\mathscr{X}$). The idea of a tensor rank was introduced by Hitchcock [8] as he proposed, the polyadic form which is in essence the representation of a tensor as a sum of finite number of simple rank-1 tensors. It was later adopted by many mathematicians and used in the CANDECOM [9]/PARAFAC [10] or CP decomposition. Rank($\mathscr{X}$) can also be seen as the smallest number of components of an exact (equality) CP decomposition.
A tensor $\mathscr{A}$ of order N has rank 1 if it equals the outer product of N of vectors $v^{(1)}, v^{(2)}, \cdots, v^{(N)}$, or for all values of i, we have:

$$a_{i_2 i_2 \cdots i_N} = v_{i_1}^{(1)} v_{i_2}^{(2)} \cdots v_{i_N}^{(N)} \tag{16}$$

There is however a difference between a rank of a tensor and that of a matrix, the main one being that the rank($\mathscr{X}$) can be different in $\mathbb{R}$ and in $\mathbb{C}$. Further information can be found at Kruskal [11].

3.1.2 Computing CP Decomposition

One of the biggest problems with CP is that there is no finite method to compute the rank.Therefore it is difficult to choose the correct number of rank one components. Most methods just try many different CP decompositions until one is found which is good, according to some measure. If the data is noise free, one could try different numbers for R iteratively until a fit of 100% is obtained.

On the contrary if the data is noisy then the fit alone cannot determine the rank. We refer the interested reader to Bro and Kiers [12] where they propose a method to test for different number of components.

There are many algorithms that can compute a CP decomposition given a fixed number of components. Here we will describe one of the most used ones using Alternating Least Squares.

For a 3rd order tensor $\mathscr{X} \in \mathbb{R}^{I\times J\times K}$ we want to compute a CP decomposition that minimizes the error (in the least squares sense) between our decomposed $\mathscr{X}$ and the original one.

We then have

$$\min_{\hat{\mathscr{X}}} \|\mathscr{X} - \hat{\mathscr{X}}\| \text{ with } \hat{\mathscr{X}} = \sum_{r=1}^{R} \lambda_r a_r \circ b_r \circ c_r = [\![\lambda; A, B, C]\!] \tag{17}$$

In the ALS method we want to fix all the matrices except for one. For example, we fix B and C and solve for A. And so on for the other matrices. When we do this, we obtain a linear least squares problem at each step. We do this until some convergence criterion is satisfied

Then we obtain the following optimization problem:

$$\min_{\hat{A}} \| X_{(1)} - \hat{A}(C \odot B)^T \|_F \tag{18}$$

Where $\hat{A} = A \cdot diag(\lambda)$.

We are trying to find the decomposition(where B and C are fixed) that minimizes the distance to our original mode 1 matrix. Then we can solve for $\hat{A}$.

The optimal solution is then given by:

$$\hat{A} = X_{(1)}[(C \odot B)^T]^{\dagger} \tag{19}$$

Where † refers to the pseudo-inverse

Finally, we normalize the columns of $\hat{A}$ to get A. Meaning we let $\lambda_r = \|\hat{a}_r\|$ and $a_r = \hat{a}/\lambda_r$. The same procedure is done for the other matrices.

In the implementation we used a library called Tensorly [13]. It is really easy to implement. Below is a snippet of CP Decomposition,utilizing the function parafac().

```
'''
Input:
    tensor:ndarray
    rank:int Number of components.
Returns:
    weights:1D array of shape (rank, ). All ones if normalize_factors is False (default),
    weights of the (normalized) factors otherwise.
    factors: List of factors of the CP decomposition, element i is of shape
    (tensor.shape[i], rank)

'''
from tensorly.decomposition import parafac
factors = parafac(tensor, rank=2)
```

Algorithm 1: Rank 2 CP Decompostion

3.2 Tucker Decomposition

The Tucker decomposition is a form of higher-order PCA. It decomposes a tensor into a core tensor multiplied by a matrix along each mode, as we can see in figure 4. In the three-way case where $\mathscr{X} \in \mathbb{R}^{I\times J\times K}$, we have:

$$\mathscr{X} \approx \mathscr{G} \times_1 A \times_2 B \times_3 C = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r = [\![\mathscr{G}; A, B, C]\!] \tag{20}$$

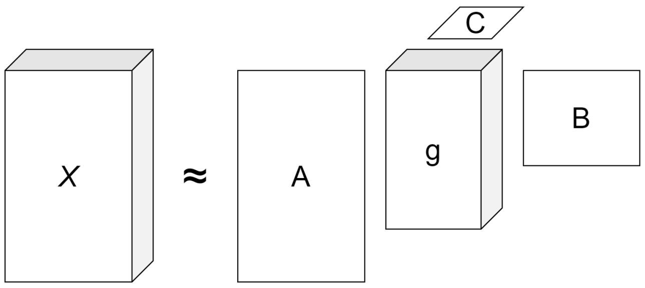

Fig. 4: Tucker Decomposition. Source: [2]

Here, $A \in \mathbb{R}^{I \times P}$,$B \in \mathbb{R}^{J \times Q}$, and $C \in \mathbb{R}^{K \times R}$ are the factor matrices (which are usually orthogonal) and can be thought of as the principal components in each mode. The tensor $\mathscr{G} \in \mathbb{R}^{R \times Q \times R}$ is called the core tensor and its entries show the level of interaction between the different components.

Here P, Q, and R are the number of components (i.e., columns) in the factor matrices A, B, and C, respectively. If P,Q,R are smaller than I, J,K, the core tensor $\mathscr{G}$ can be thought of as a compressed version of $\mathscr{X}$. Several notions on the rank of a tensor carry over to that of its independent Tucker core, this can be seen at [14].

3.2.1 n-Rank

Denoted by $\text{rank}_n(\mathscr{X})$, the n-rank was introduced by Kruskal [11]and it is important in order to understand Tucker decomposition which is also discussed in the following section. it is should be noted that it is different from rank-1 components of a Tensor.
For $\mathscr{X}$, an N$^{\text{th}}$-order tensor sized $I_1 \times I_2 \times \cdots \times I_N$, n-rank is the dimension of the vector space spanned by the mode-n fibers of $\mathscr{X}$ or, $\text{rank}_n(\mathscr{X})$ is the column-wise rank of $X_{(n)}$. Moreover,if $\text{R}_n = \text{rank}_n(\mathscr{X})$, we can infer that $\mathscr{X}$ is a tensor of rank $(\text{R}_1, \text{R}_2, \cdots, \text{R}_N)$ and we can find an exact Tucker decomposition of this rank. However, if $\text{R}_n < \text{rank}_n(\mathscr{X})$, for at least one n, then this rank's Tucker decomposition will be inexact and difficult to compute.

3.2.2 Computing Tucker Decomposition

The basic idea for computing Tucker Decomposition is to find the components that best capture the variation in mode n , independent of the other modes. This was the first method introduced by Tucker [15] and it is better known as the Higher Order SVD. If we choose R_n(the desired rank) to be less than the Rank_n of $\mathscr{X}$ for one or more n it is called the truncated HOSVD.

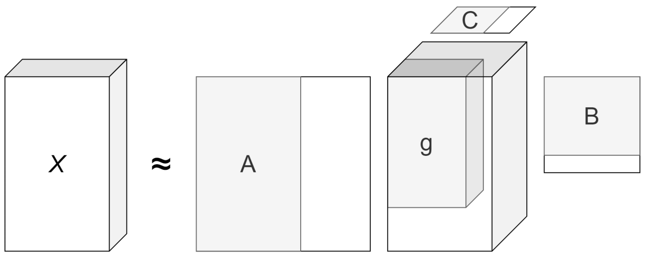

Fig. 5: Truncated HOSVD. Source: [2]

The truncated HOSVD is not optimal in the sense that it doesn't give the best fit when measured using the norm difference. But it is a good starting point for Iterative ALS algorithms. One of the methods that improves on HOSVD is the Higher Order Orthogonal Iteration (HOOI). This method uses more efficient techniques for calculating the factor matrices. It computes only the dominant singular vector of X_n and uses a SVD rather than eigenvalue decomposition.

The optimization problem of the HOSVD is as follows

$$\begin{aligned} \underset{\mathcal{G}, A^{(1)}, \dots, A^{(N)}}{\text{minimize}} \quad & \| \mathcal{X} - [\![\mathcal{G}; A^{(1)}, A^{(2)}, \dots, A^{(N)}]\!] \| \\ \text{subject to} \quad & \mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times, \dots, \times R_N}, \\ & A^{(n)} \in \mathbb{R}^{I_n \times R_n} \qquad \text{and columnwise orthogonal for } n = 1, \dots, N. \end{aligned} \tag{21}$$

Which is then solved using the Alternating Least Squares Method.

Tucker decomposition is easily implemented in Tensorly by specifying the rank along each mode.

```python

from tensorly.decomposition import tucker
'''
Input:
    tensor:ndarray
    rank: None, int or int list.If int, the same rank is used for all modes.
Returns:
    core:ndarray of size ranks. Core tensor of the Tucker decomposition
    factors: ndarray list. list of factors of the Tucker decomposition.
    Its i-th element is of shape (tensor.shape[i], ranks[i])

'''
core, factors = tucker(tensor, ranks=[2, 3])
```

Algorithm 2: Tucker Decomposition

4 Representing videos as tensor

Before jumping directly into code and tensor calculus exploits, it's appropriate to explore videos and it's properties. This section provides a brief summary on video structures. All the information provided over this section is a simplification from [16] and [17] which provide plenty information on video coding.

4.1 What's a video?

There are mainly two types of videos, **Analog** and **Digital** *videos.*

- Analog video
  - These are generated as an output of video camera's sensors through a process called scanning. In simplified terms, the scene is represented as a **2-D** projection on a **1-D** electrical signal that on most cameras starts from the bottom left of the sensor and ends at the bottom right of it. Signals are generated corresponding to the color space of the video.
  - The video scene is conceived by rapidly generating the same 2-D projections one after the other and giving the illusion of movement from the differences of one picture after the other.
- Digital Video

  - Digital video requires the transformation from analog video into a digital storage format. This process is done by applying certain filtering, sampling and quantisation of the analog video signal.
  - On principle, digital video is also a sufficient quantity of 2-D projections played on a determined time interval.

In general terms, digital video is generated from a conversion of the analog signals generated by a camera or from any kind of digital animation software that generates this from scratch. Nowadays, digital video is the most common representation by far, and analog video is a technology that is mostly just part of cameras and is not perceptible in further uses. Therefore, in the following sections we will focus only on digital videos as videos.

4.2 Digital Video properties

In general, videos are *4-Dimension* objects (color space, height, length and frames). With the defined digital videos structures, videos are represented on computers. A quick overview of this properties on this section should provide an understanding on videos mechanics.

4.2.1 Color Spaces

Videos rely on defined techniques to display color images. These are known as color spaces and many have been designed through history to targeting specific goals for and moving pictures. For our purposes, this are worth mentioning:

- Black and White
  - This is basically any video without colors (In a non-strict sense of the word color or shade). In this, each pixel requires only one binary number to represent brightness (0 for black and 1 for white). Concretely this is the lightest way to represent an image.
- RGB (BGR)
  - This is probably one of the most famous, used and easier to understand color spaces. RGB stands for the three primary additive colors of light, *Red, Green and Blue.* Basically with this approach any color can be generated using specific combinations of each.
  - All the color information is stored in channels for each color and have the same domain and importance.
  - Figure 6 is a RGB decomposition on how the superposed combination generates a full color bitmap.

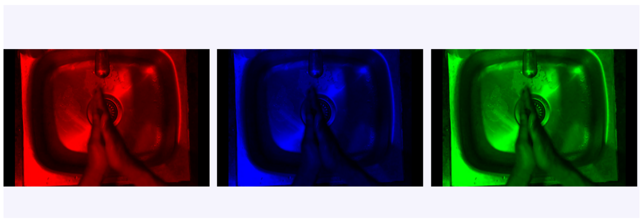

Fig. 6: Representation over RGB decomposition channels. Source: [18]

- Gray scale
  - This is the closest to Black and white in a visual sense. This is basically a projection using luminance information in shades of grays usually converted from an RGB or Y:C_r:C_b color space into a **1-D** value.

For each color space integers for the combinations of additives channels are defined as color depth. Each channel has a domain defined by multiples of 2-bits. For example, black and white is represented by 2-bits (0 for black and 1 for white). Gray scale is represented with at least 4-bits and one integer, and the rests of color spaces are represented with usually 8-bits or 10-bits and 3 integers, giving a total of 255 possible shades per channel. 12-bits or 16-bits variances of raw video can be found, but it's certainly not common and demands one degree more of resources than usual.

4.2.2 Frames

Another important property of videos are frames. There are mainly two things to take in consideration from them, Progressive Sampling and Interlaced Sampling. This properties can be interesting at the time of studying motion on videos.

The quantity of frames per seconds (FPS) is other of the important properties that we will need later. This one varies from sources and standards, but for standard videos intended to be visualized by humans it is minimum 24fps. Certain applications lower rates around 10-20fps are used as low bit-rated applications (Like camera surveillance systems).

4.2.3 Resolution

A pixel (px) is the smallest distinguishable element in an image. In another sense is the smallest distinguishable variation from a video signal sequence [19]. Each pixel is used to represent a color point by a combination of the information of the channels from the color space.

For our purpose, this pixels are used as coordinates on a fixed-rectangular-grid. This grid has a 2-dimensions (height and length) that defines the resolution of the video. If certainly a grid of any size can be defined, there exist standards that are pretty common. As today, the most common format of videos are 720p ($1280px \times 720px$), 1080p ($1920px \times 1080px$), generated by basically any electronic device. Of course, smaller or bigger resolution grids such as SD ($720px \times 576px$) or 4K ($3840px \times 2160px$), where the later is the one being more adopted everyday.

4.3 Information density

So far 3 intrinsically properties from videos have been defined. With this information, a description of the complexity of videos as data can be prescribed. As mentioned before, there exist standards formats that have fixed values for each of this properties. The examples used in this paper are of videos of 30fps, a resolution of $720px \times 480px$, and unloaded as 8-bits RGB. Simply by multiplying $720px \times 480px \times 30fps = 10,368,000px/s$. This is the main reason uncompressed video is not as common and the importance of compression for storage and transmission.

4.3.1 Videos representation as tensors

With the defined framework, it is possible to shape videos into a mathematical object to work with. Tensors being a multidimensional array, each of the previously mentioned video properties can be linked to a certain dimension and therefore be represented as such. Respectively *Color Space, Frame Rate and Resolution* properties from videos are the ones of interest. Figure 7 shows a representation of the conversion from videos to a tensor object by organizing each of the previously mentioned properties as one of the tensor dimensions.

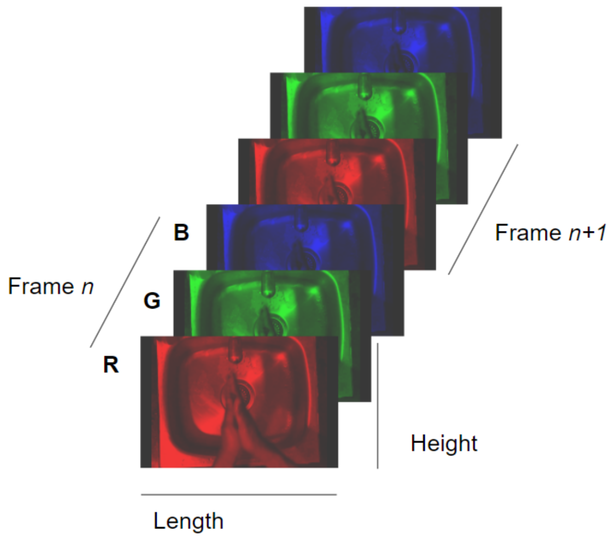

Fig. 7: Representation of videos as a Tensor. Source: [18]

On practice, this conversion can be easily implemented. In this paper the libraries Open Computer Vision [20] (**CV2**) and **Tensorly** [13] are of special interest. **CV2** provides tools for capturing frames from video files and making color spaces conversions easily. **Tensorly** helps in the conversion of this read frames into a tensor structure, offering as mentioned before tools to manipulate and decompose this videos.

Other papers have explored invariant or best approximation of tensors norms and eigen-decomposition by using the core from the Tucker Tensor Decomposition (See [14]) and they can be used to represent the original videos.

It's well known that computing tucker-decomposition is a rather costly operation. Nevertheless, this decomposition still provide a significant advantage since they can be stored and called from disk. Depending on the n-rank used, this decompositions can be of such small scale compared to the original file, that it becomes trivial to store. In another hand, with nowadays hardware tools like GPU or TPU, the calculations required for the decompositions can be done significantly quicker. For this, **Tensorly** provides options to use different back-ends. If the computer to be used has a **CUDA** compatible GPU, back-ends like **Torch** [21] or **TensorFlow** can be used to improve computation time, otherwise a slower but still useful **numpy** [22] back-end can be used.

5 K-means Clustering

Since we wanted to compare tensors, we decided on using a clustering algorithm. In comparison to using a distance measure(which just compares two tensors), a clustering algorithm allows us to compare several tensors and group them based on their similarity.

Before we introduce the k-means method we need to define some concepts:

Clustering: It's the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other clusters.

Quality Criterion: It is usually used to stop the iterations after some threshold is met. Typically a sum of squared errors is used.

The k-means method is a clustering technique that is used to minimize the average squared distance between points in the same cluster.

The first k-means Algorithm was proposed by Lloyd [23].It first starts by choosing, usually at random, k arbitrary centers from the data. Each data point is then assigned to the nearest center, and each center is recomputed as the center of mass of all points assigned to it. These steps are repeated until a quality criterion is met.

5.1 Algorithm

Input: Example set B, number of Clusters k, convergence parameter ϵ

```
Select examples randomly and assign them to $m_1,...,m_k$
$Q_{new} := \infty$
FOR $i = 1$ to $k$
    DO: $C_i := \{\}$
REPEAT
    FOR $b \in B$
        DO: assign b to a cluster $C_i$ such that $d(m_i, b)$ is minimal
    FOR $i = 1$ to $k$
        DO: $m_i := \frac{1}{|C_i|} \cdot \sum_{b \in C_i} b$
    $Q_{old} := Q_{new}$
    $Q_{new} := \sum_{i=1}^{k} \sum_{b \in C_i} d(m_i, b)^2$,
    UNTIL $Q_{old} - Q_{new} < \epsilon$
```

Here d is the euclidean distance, Q is the quality measure and m_i is the center of each cluster

5.2 Performance Evaluation

With any kind of algorithm, there remains the question if the results provided are sufficient. For clustering algorithms there exist roughly two types of performance evaluation that depend if a ***ground-truth*** is known or not. On the data-set used in our experiments section 6.3, there exists predefined labels describing the ground-truth. Since k-means results depend on the starting points, results could be biased due to chance and the selected performance evaluation measure must take this into account.

There's a great guide for performance evaluation provided by the **scikit-learn** [24] website which was used to select a good scoring technique and also during the experiment implementation [25]. With the defined requirements the *Adjusted Mutual Information Score* described over [26] probed to be the best fit. *Mutual Information* methods are well known, and the adjusted version is a more recent iteration correcting for chance.

Results from it can be interpreted by the obtained value for $AMI \in [-1, 1]$ interval:

- $AMI = 0$ the relation of the clusters and labels is understood to be random.
- $AMI > 0$ indicate significant agreement, where $AMI = 1$ means that the cluster and label assignment are equal.
- $AMI < 0$ indicates independent clustering and labeling relation. Scores are to be carefully evaluated and interpretation depends of the field of application.

6 Experiment

The aim of these experiment is to test if using tensor decompositions is a good way for identifying differences between videos. More specifically, examining if the core tensors are a good compressed representation of the original video. Also exploring the efficiency improvement of calculations over a lower dimensional space in comparison to using the complete video tensor.

6.1 Hypothesis

In our data-set we have mainly two inherent groups/settings for videos (step-number and sink-type). Our hypothesis is: *Videos with the same label should be similar.* In other words we expect that videos with the same label end up in the same clusters.

6.2 Data-set

For the purpose of analysis, we picked a standard data-set, the Hand Wash Data-set [18] from Kaggle. The data-set consists of 300 individual videos of hand washes with each hand wash having 12 steps (fig.8), in different environments to provide as much variance as possible with respect to: illumination, background, source camera position, field of view and individuals performing the hand wash.

The Hand Wash Data set intends to simulate the real-world restrictions from action recognition applications, such as: fixed camera position, real-time feedback, varying illumination, static background and applied to one domain-specific fine-grained action task.

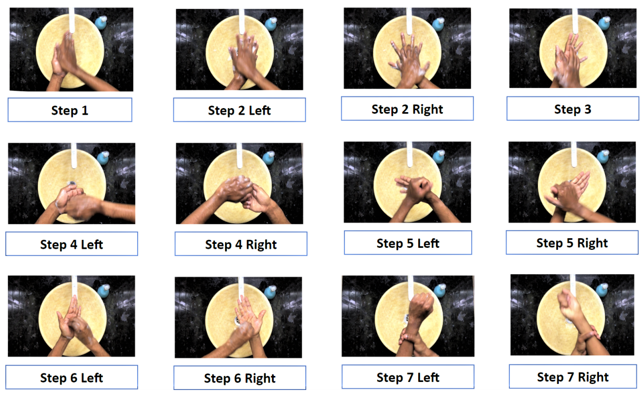

Fig. 8: Example for steps from the Hand Washing Data Set. Source: [18]

The data set is further decomposed into 6 settings describing the type of sink, them being *DarkSteelSink1, LightSteelSink1, CeramicSink1, SmallCeramicSink6, LightGraniteSink1, CeramicSink2*. As seen below in fig. 9

6.3 Methodology

The approach for the implementation part of these research was trying to reproduce the results from and experiment that can be found at [27].

The idea is converting a video into a lower dimensional space to allow comparisons between videos. This is done by performing Tucker decomposition. The core tensor is then used as a compressed version of the video.

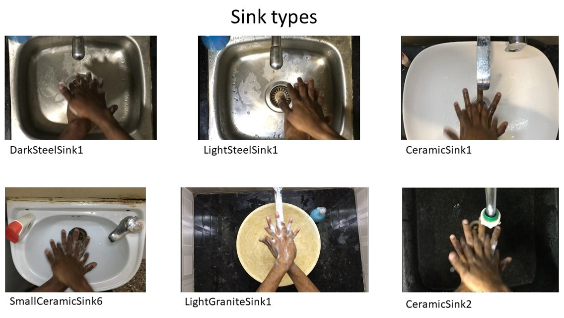

Fig. 9: Example for type of sink from the Hand Washing Data Set. Source: [18]

For comparing the similarity between two given core tensors, the L2 norm of the difference between the two core tensors was used.

The L2 norm of a tensor $\mathscr{X}$ is given by:

$$\|\mathscr{X}\| = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \dots \sum_{i_n=1}^{I_N} x^2_{i_1 i_2 \dots i_N S}} \tag{22}$$

In the mentioned work only 3 videos were used, therefore the results were not significant. We wanted to expand on the idea and use a bigger data set, to be able to infer if this approach is useful.

Since our approach involved a much bigger data set (75 videos in total) we had to refactor the code to make it more efficient. Our code can be found in our GitLab repository given at [25]:

For simplification we took a subset of the steps to analyze and computed the Tucker Decomposition. From the 12 steps we selected the following: Step 1 (Represented as 1), Step 2 Left (Represented as 2), Step 3 (Represented as 4).

All the videos from this steps were considered for the current analysis.

We mainly tried two different decompositions. The first using the full color videos and decomposing them using $rank_n = (10, 10, 10, 2)$ and 100 frames. The second using a gray scaled version an computing a decomposition with $rank_n = (32, 32, 32)$ and 200 frames.

The approach for the gray scale version is the following:

1. Transform the video to gray scale, select 200 frames at random and perform a tucker decomposition using $rank_n = (32, 32, 32)$. This number was used because it was at the limit of our computational capabilities. The decomposition's are then stored as objects and saved in files. Each object file contains the core and factor matrices for an specific video. We then repeat this step for all 75 videos.

2. Perform a k-means algorithm, grouping similar videos into clusters. For *Sink-type* $k = 6$ and for *Step-number* $k = 3$.

3. Summarize results for the current configuration.

The same steps are then performed for the full color version, without transforming the videos into gray scale.

6.4 Results

Over the implementation on GitLab [25], a fixed seed had been used to enable reproducibility to satisfy any type of further curiosity. After defining the K-Means experiment over section 6.3 and it's performance evaluation on section 5.2 an appropriate performance evaluation for this experiment, we ran a total of **500 iterations** in order to: compute an average AMI, get an idea of how many iterations for convergence were required and visualize how the clusters were organized for our best iteration. Table 1 contains the acquired results:

Feature	Sink-Type		Step	
Configuration	Gray-scale	RGB	Gray-scale	RGB
Average quantity of steps for convergence	4.54	4.16	3.3	3.48
Average AMI	0.42	0.44	0.0	0.06
Best AMI	0.56	0.58	0.05	0.09

Tab. 1: Results table

Summarizing, an score $AMI \approx 0.4$ was obtained with respect to the *Sink-Type*, certainly showing an agreement between clusters and the true-labels. Furthermore, an $AMI \approx 0.0$ with respect to the *Step-number* can be interpreted that the relation between the clusters and the true-labels is of a random behavior. It must be also noted that each k-means run converged successfully on around 4-5 iterations.

It can help to also visualize the relationship between cluster and label on a table grid. The figures 10 and 11 show the results for each configuration and features used for the experiment and it's margins (The sums per clusters and per labels). A good agreement between clusters and labels can be represented with an sparse matrix. When $AMI = 1$ should indicate a permuted diagonal matrix. On the other hand, really dense matrices represent randomness or no mutual agreement ($AMI \leq 0$). *Note that we used heat-maps to highlight the sparsity or not of these tables. The temperature interval was set from 0 (Black) to the minimum value from the margin axis (Beige).*

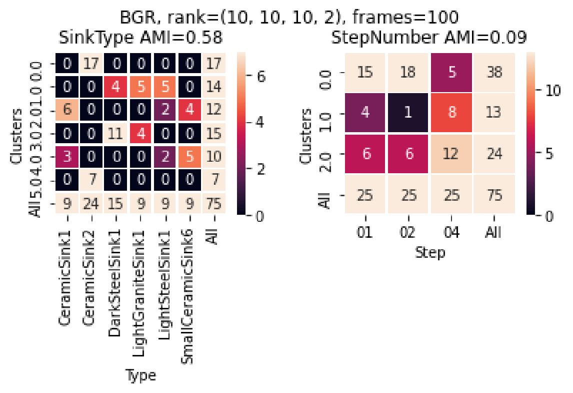

Fig. 10: Results for RGB configuration

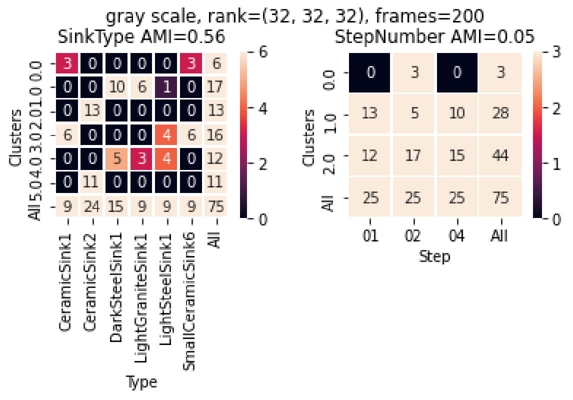

Fig. 11: Results for gray scale configuration

7 Conclusions

Our proposed hypothesis says that similar videos should be part of the same clusters. This hypothesis was tested for the Step-number or Sink-type configuration separately.

From the results on table 1 and the summary over section 6.4 it can be said that:

- It was not possible to define similarity between videos from the same step.
- There exist a similarity between videos with the same type of sink.

With our configurations the resulted tucker cores were able capture the static nature of the videos and their backgrounds (Attributed to the type of sink). Even using such simple algorithms like k-means, the cores approximated the videos essence while using a lower dimension space, requiring therefore less computations and storage space than the one from the original data.

7.1 Remarks

During the experimentation phase, several different settings had been implemented, which did not improve on the previous results. These are listed below:

- The use of a bigger rank did not give a substantial improvement on the results.The average AMI obtained with a $rank_n = (2,2,2,2)$ was 0.39, compared to the 0.44 obtained with a $rank_n = (10,10,10,2)$. More over any improvement must be weighted or compared against the computational time, since tucker decomposition is a really machine intensive algorithm.
- Increasing the number of iterations did not provided significantly different results. In fact, testing runs with less than *50 iterations* showed the same results.
- The difference between RGB and Gray-Scale is almost negligible. For example, in the Sink-Type experiment the difference in the average AMI is 0.01

7.2 Further work/ Next steps

All in all, despite adopting a somewhat rudimentary approach in this paper, we were able to get acceptable results. Nevertheless, it is clear that for real-world applications further analysis should be carried out. There are many things that we listed as next steps to keep developing basis around using videos as tensors:

We can pair what we already have (for the k-means algorithm) with other distance measure such as Chebyshev, Manhattan or Cosine Distance or similarity measures such as Pearson's correlation or correlation similarity or even using different clustering techniques.

Furthermore, using appropriate algorithms that take into account the dynamic nature of the videos, for which a simple tucker and distance measure may not suffice. Gao, Yang, Tao and Li [28] propose a novel approach for video semantic analysis. With two main components, optical flow tensors and hidden Markov models, they employ motion information of a video to detect the semantic information embedded in the video and preserve the information structure of the video throughout the dimensionality reduction. They also perform general tensor discriminant analysis and linear discriminant analysis for dimensionality reduction. Results show feasibility and promise.

Also exploring Xu, Khan, Zhu, Han, K. Ng and Yan [29] suggestion of a tensor-based large margin discriminative framework for visual tracking using supervised tensor learning. They produce a multi-linear decision function using an online support tensor classifier utilizing the nonlinear tensor-based features over the target. Simultaneously, using a truncated Tucker decomposition to produce a superior parameter tensor reconstruction while updating the classifier.

References

1. WANG, C.-C.; BOWEN, R. M.: *Introduction to Vectors and Tensors: Linear and Multilinear Algebra*. 1976. ISBN 0306375087.
2. KOLDA, T. G.; BADER, B. W.: Tensor Decompositions and Applications. *SIAM REVIEW*. 2009, vol. 51, no. 3, pp. 455–500.
3. S. PARK Suan Lee, J. K.: Time-Sensitive Multi-Dimensional Recommender in database system. 2017.
4. BADER, B. W.; KOLDA, T. G.: Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*. 2006, vol. 32, no. 2, pp. 635–635.
5. DE LATHAUWER, L.; DE MOOR, B.; VANDEWALLE, J.: A multilinear singular value decomposition. *SIAM Journal of Matrix Analysis and Applications*. 2000, vol. 21, no. 4, pp. 1253–1278.
6. A. SMILDE, R. B.; GELADI, P.: *Multi-way Analysis: Applications in the Chemical Sciences*. 2004. ISBN 978-0-471-98691-1.
7. LOAN, C. F. V.: The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*. 2000, vol. 123, no. 1-2, pp. 85–100.
8. HITCHCOCK, F. L.: The Expression of a Tensor or a Polyadic as a Sum of Products. *Studies in Applied Mathematics*. 1927, vol. 6, no. 1-4, pp. 164–189.
9. CARROLL, J. D.; CHANG, J.-J.: Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*. 1970, vol. 35, pp. 283–319.
10. HARSHMAN, R. A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*. 1970, vol. 16, pp. 1–84.
11. KRUSKA, J. B.: Rank, decomposition, and uniqueness for 3-way and N-way arrays. 1989, pp. 115–121.
12. BRO, R.; KIERS, H. A. L.: A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics*. 2003, vol. 17, pp. 274–286.
13. KOSSAIFI, J. et al.: TensorLy: Tensor Learning in Python. *Journal of Machine Learning Research (JMLR)*. 2019.
14. JIANG, B.; YANG, F.; ZHANG, S.: *Tensor and Its Tucker Core: the Invariance Relationships*. 2016. Available from arXiv: 1601.01469 [math.OC].
15. TUCKER, L. R.: Some mathematical notes on three-mode factor analysis. *Psychometrika*. 1966, vol. 31, pp. 279–311.
16. RICHARDSON, I. E.: *The H.264 Advanced Video Compression Standard*. 2003. ISBN 9780470516928.
17. GHANBARI, M.: *Standard Codecs: Image Compression to Advanced Video Coding*. 2003. ISBN 9781849191135.
18. WHO; REAL-TIMEAR: *Sample: Hand Wash Data-set* [https://www.kaggle.com/realtimear/hand-wash-dataset/metadata]. Kaggle, 2019.
19. GRAF, R. F.: *Modern Dictionary of Electronics*. 1999. ISBN 0750643315.

20. OPENCV: *Open Source Computer Vision Library*. 2015.
21. COLLOBERT, R.; KAVUKCUOGLU, K.; FARABET, C.: *BigLearn, NIPS Workshop*. Torch7: A Matlab-like Environment for Machine Learning. 2011.
22. HARRIS, C. R. et al.: Array programming with NumPy. *Nature*. 2020, vol. 585, no. 7825, pp. 357–362. Available from DOI: 10.1038/s41586-020-2649-2.
23. LOYD, S.: Least squares quantization in PCM. *Technical Report RR-5497, Bell Lab*. 1957.
24. PEDREGOSA, F. et al.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830. Available also from: https://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation.
25. ELMORSY, A.; FERNÁNDEZ HERRERA, J. C.; JÁQUEZ LORA, J.: *Explore tensors in videos* [https://gitlab.uni-trier.de/s4jnfern/explore-tensors-in-videos]. Universität Trier GitLab, 2021.
26. VINH, N. X.; EPPS, J.; BAILEY, J.: Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary? In: *Proceedings of the 26th Annual International Conference on Machine Learning*. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 1073–1080. ICML '09. ISBN 9781605585161. Available from DOI: 10.1145/1553374.1553511.
27. HERMES, C.: *Video Analysis with Tensor Decomposition in Python* [https://towardsdatascience.com/video-analysis-with-tensor-decomposition-in-python-3a1fe088831c]. Towards Data Science, 2019.
28. A, X. G. et al.: Discriminative optical flow tensor for video semantic analysis. *Computer Vision and Image Understanding*. 2009, vol. 113, pp. 372–383.
29. AB, G. X. et al.: Discriminative tracking via supervised tensor learning. *Neurocomputing*. 2018, vol. 315, no. 2, pp. 33–47.

List of Figures

List of Tables