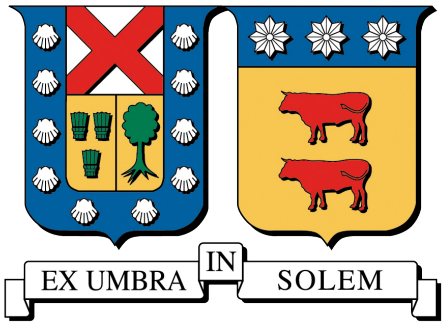
2021-06

CONVOLUTIONAL NEURAL NETWORK FEATURE EXTRACTION USING COVARIANCE TENSOR DECOMPOSITIC

FONSECA ROMERO, RICARDO ESTEBAN

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
Departamento de Electrónica



Convolutional Neural Network Feature Extraction using Covariance Tensor Decomposition

Ricardo Esteban Fonseca Romero

TESIS DE TITULACIÓN PARA OPTAR AL GRADO DE DOCTOR EN INGENIERÍA ELECTRÓNICA

PROFESOR GUÍA : Dr. Werner Creixell

JUNIO 2021

TÍTULO DE LA TESIS:

Convolutional Neural Network Feature Extraction using Covariance Tensor Decomposition

AUTOR:

Ricardo Esteban Fonseca Romero

Trabajo de tesis presentado en cumplimiento parcial de los requisitos para el título de Doctor en Ingeniería Electrónica de la Universidad Técnica Federico Santa María.

Dr. Werner Creixell _____

Dr. Cesar Caiafa _____

Dr. Moulay Akhloufi _____

Dr. Mauricio Araya _____

Valparaíso, Junio de 2021.

ACKNOWLEDGMENT

This work was supported by the Grant Comisión Nacional de Investigación Científica y Tecnológica (CONICYT) Programa Formación Capital Humano Avanzado (PCHA) Doctorado Nacional 2016, Chile, under Grant 21161259, in part by the Universidad Técnica Federico Santa María (UTFSM) Project, Chile, under Grant 116.23.3, and in part by Digevo, Chile, Cloud computing by Nvidia Inception and Amazon Web Services (AWS) Activate.

This work was published in IEEE Access journal [17].

Contents

List of Figures

List of Tables

1 Definition of Thesis Topic

1.1 Title

Convolutional Neural Network Feature Extraction using Covariance Tensor Decomposition

1.2 Abstract

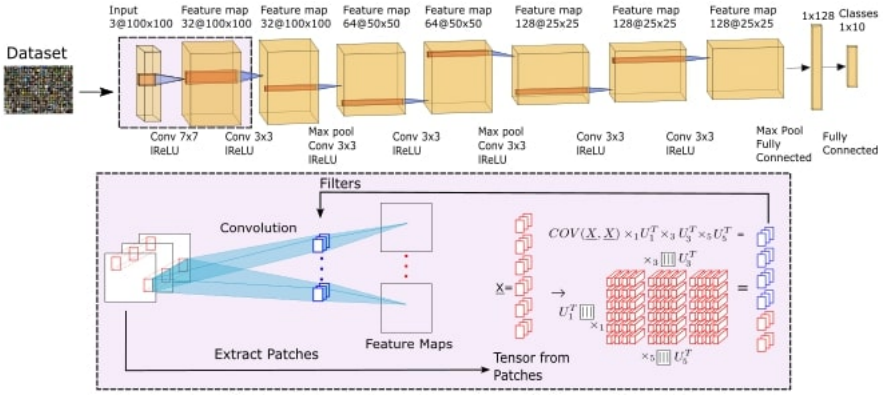


Figure 1: VGG 7 classification architecture and Covariance Tensor to generate image features.

This work describes a new method to extract image features using tensor decomposition to model data. Given a set of sample images, we extract patches from images, compute the covariance tensor for all patches, decompose with the Tucker model, and obtain the most critical features from a tensor core. To extract features, we factorize the covariance tensor (CovTen) into its core and propose a new interpretation of the resultant tensor structure, which holds relevant features in a block-wise arrangement (also named filters, weights, or kernels). This tensorial representation allows preserving the spatial structure, learning multichannel filters, and establishing linear dependence between dimensions, reducing the dimensional complexity (the curse of dimensionality). Thus, the proposed method generates filters by a single feed-forward step using a few samples per class as low as 1. Besides, in kernel generation, labels are not needed. The obtained features were extensively tested using a convolutional neural network for classification. All tests were conducted under the VGG architecture conventions. The experiments helped us identify the proposed method's advantages versus traditional convolutional neural networks in inference capacity and kernels initialization. We also performed experiments to select hyperparameters (nonLinearity, max pooling, samples, filter size) according to their performance. The inference capacity results showed an increased classification accuracy around 67% with CIFAR 10, 64% with CIFAR 100, and 98% with MNIST, using 10,100,1000 samples with a single feed-forward training. On the other hand, the initialization experiments showed the feature extraction capability versus available initializers (He random, He uniform, Glorot, random), confirming linear tensor constraints' usefulness to generate features. Using the method as kernel initializer returns comparable findings with state of the art around 91% with CIFAR 10, 72% with CIFAR 100, and 99% with MNIST.

2 Background

This work introduces a new algorithm based on multilinear algebra for feature extraction, which later is plugged into a Convolutional Neural Network to perform classification. We seek to evaluate the discriminative capability of the generated features and the performance as a kernel initializer. Besides, Multilinear methods have been studied since the 19th century by Kronecker, Gauss, Cayley, Weyl, and Hilbert's contributions. A detailed historical note can be found on [11]. Moreover, tensor decompositions for signal processing have been applied since 1990 with the Tucker and PARAFAC decompositions [56], [20]. A tensor is a higher-order array, where the order represents the dimension of the array and the index number accordingly. A dimension might represent spatial location, channel, time, frequency, samples, classes, or other features. Figure 2 shows some tensor examples of three, four, and five order respectively subfigures show some fibers in each mode. Although multilinear techniques are considered breakthrough advances in some areas, such methods have not been widely studied for deep learning applications.

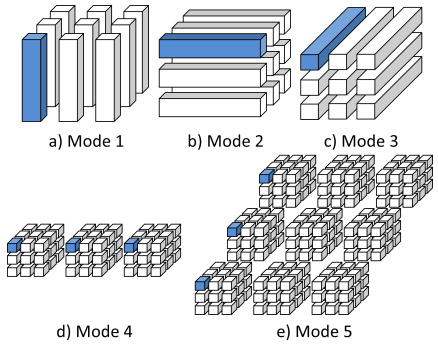


Figure 2: Different high order arrays examples with some fibers per mode.

Deep learning focuses on discovering multiple representation levels, hoping that features represent more abstract semantics of the data [10]. The features could be corners, edges, lines, regions, and others [54]. In recent years, within this field, computer vision has reached many advances, particularly with convolutional neural networks (`ConvNet`) [34], [36]. ConvNets are among the most famous deep neural networks and work with the mathematical matrix operation called convolution [34]. Its most common architecture is composed of convolutional, non-linearity, pooling, and fully connected layers. ConvNets are applied to classification, object detection, tracking by detection, counting, and other tasks. These tasks have relevant importance by their applications in several areas of knowledge. Besides, development in ConvNets is an active area, where the ImageNet contest comprises the most relevant results in visual computing contemplating classification and detection tasks [2].

Despite the outstanding research in this area, effective learning critically depends on expertise and empirical observations for tuning parameters and architectures. In this context, an attractive alternative to enhance conventional ConvNet is to understand the effects of multilinear algebra methods to generate filters. The hypothesis is that multilinear algebra and its methods, such as Tensor Decompositions (`TD`), capture multiple interactions and couplings on data. At the same time, TD assumes linear dependency between dimensions. Thus, such decompositions can discover a hidden structure that traditional matrix methods can not obtain. This hypothesis is supported by promising results obtained on several works on different areas, which include: audio, image, and video processing, machine learning, and biomedical applications [11]. Well known

TD methods are Tucker and Canonical decomposition (CANDECOMP). Tucker decomposition [56], [15], [31] minimizes the Frobenius norm of the error when a multilinear subspace approximation is performed. On the other side, CANDECOMP [20], [9] [13] applied over a High Order Statistics tensor, is able to recover independent sources similar to the Independent Component Analysis [24], [25].

Unlike matrices, tensors are multiway arrays of data samples whose representations are typically overdetermined. Therefore, it has fewer parameters in the decomposition than the number of data entries. This approach gives us enormous flexibility in finding hidden components in data and enhancing robustness to noise [11]. This multiway data analysis lets us develop sophisticated models that capture multiple interactions and couplings instead of standard pairwise interactions. In this way, we can take benefits from high dimensional data using tensor structure. As an example, tensor representation by Tucker decomposition [11] benefits the data representation by fewer parameters, the uniqueness of decompositions, the flexibility in the choice of constraints, and the generality of components that can be identified.

It is common to preserve complex data structure such as multidimensional arrays by using tensors. In practical applications, they are managed by modern GPU hardware [47]. The GPU advances are directed to optimize tensors' computation through dedicated computing units (tensor cores) inside their architectures. Therefore, these recent approaches need multilinear-based algorithms that take the GPU hardware advantages, improving the training stage, which is the most demanded in terms of resources [43].

In our proposal, features were extracted using Multilinear Subspace Learning (`MSL`). Given a training set, Linear Subspace Learning (`LSL`) [38] attempts to find a vector to project samples and capture the most relevant information, e.g., PCA, LDA, FDA, PLS, etc. On the other hand, MSL finds a multilinear subspace to project data, i.e., project matrix samples on a linear subspace built with a matrix basis. Projection on such subspaces generates a feature vector (latent variable) with a dimension given by the basis number used.

Eigenspaces [28] is an initial work in LSL which attempts to model shape and appearance variation. They project grayscale images to a low dimensional subspace using SVD. This approach relies on linear algebra and cannot model poses, variations, nor illumination. In this process, the training images are vectorized (images to columns) and grouped on a data matrix. Then, SVD over this assembled matrix is used to compute the eigenfaces as the left singular vectors. On natural images exist spatial correlation on a neighborhood, which is eliminated by the vectorization. Finally, the extracted features are used to classify images [23].

On a dataset composed of images with size $I_1 \times I_2$, feature extraction by eigenfaces [28] leads to subspaces with $I_1 I_2$ coefficients and at most the same number of basis vectors. In general, such high dimensional sets are challenging to handle, so feature selection and dimensionality reduction techniques should be considered to represent data using fewer variables. Additionally, such estimation suffers from the curse of dimensionality: a higher number of parameters than the number of samples. For example, for a QVGA image 320×240 pixels, a subspace with 76800 coefficients should be estimated, while the number of samples on the LFW dataset lies around a few thousands. Our method was motivated by the advantages of multilinear decompositions in contrast to standard matrix counterparts. Image datasets are high-order arrays; hence a traditional flattened view will not represent data properly. This fact reflects the limitations of classical matrix models and the necessity to move toward more versatile data analysis tools [11].

Most recent approaches seek to replace the standard kernel initialization of ConvNet for a PCA-based method and propose a parametric equalization normalization to adjust

the scale among the neuron weights [10], [62]. This technique uses image samples per class from the training dataset to get more relevant information; in other words, it extracts the principal features in weighted kernels. In this way, the method effectively overcomes the uncertainty caused by the standard kernel's initialization and accelerates the training process.

In our work, during a single feed-forward step, we generate the kernels for a ConvNet architecture by computing the covariance tensor of the data and factorizing it by Tucker decomposition [65], [56]. The method starts from the findings of Kirby and Sirovich [28], who introduce the computation of eigenfaces. We also inspired our proposal in the cascade convolutional architecture procedure of PCANet [10]. Additionally, our proposed method works with a small sub dataset (which covers all the features spectrum of the classes), similar to the work of Wang et al. [62]. However, to the kernel generation, labeled data is not needed. Accordingly, the technique is applied to study two main tasks: inference classification (capacity) and kernel initialization, compared with other already well-known methods.

Moreover, to measure the performance, we employ a VGG ConvNet architecture [50]. This work essentially remarks on the feature extraction capabilities of our method. Nevertheless, this work also pretends to be a good baseline for future research incorporating this method during the training stage.

2.1 Problem identification

- How to extract features from a set of sample images preserving matrix and tensor structure?
- Given information of class labels. How to find features that best relate image samples to labels?
- Linear models able to relate features and labels fail if the relation is nonlinear. How to exploit hidden non linearities and determine their utility?
- Eigenfaces and tensorfaces led to estimation of a high number of parameters per feature. How to extract features with less number of parameters?

3 Linear Subspace Learning

In this section we discuss the principal tools and traditional approaches to represent data through linear transformations. The goal is to exploit or extract latent information on the data, principally exploiting data statistics, i.e. moments and cumulants. This task is known as **Linear Subspace Learning** (LSL).

3.1 Principal Component Analysis

Consider the projection problem, where we are given a set of m points on a n-dimensional space $\left[x^{(1)}, \cdots, x^{(m)}\right] = X$, $x^{(i)} \in \mathbb{R}^n$. The goal is to find a unitary vector v such that maximize the magnitude of the projection $\sum_{i=1}^{m} v^T x^{(i)}$ over the m samples. The projection $v^T x^{(i)} = \left\langle v, x^{(i)} \right\rangle$ is the inner product between each sample (ith column from X) and the vector v. This problem is also known as to find the vector v that the variance of the projected data is maximized.

Graphically this problem was depicted on figure 3. Consider m samples of a two dimensional random vector $x^{(i)} \in \mathbb{R}^2$ drawing according to a jointly probability distribution function $x \sim \mathcal{N}(0, diag\left[\sigma_1, \sigma_2\right])$. Samples from this multivariate random variable were plotted on figure 3.

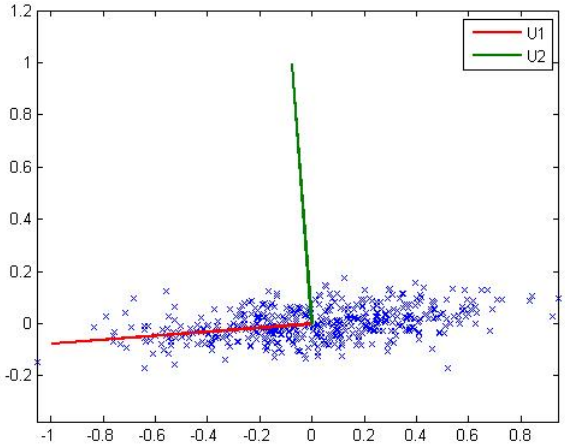


Figure 3: Data points and principal components

Consider the projections of all samples on the vector v as $v^T X$. As we are interested on the magnitude, the square of the projection $v^T X X^T v$ was used. The former quadratic term is our objective function and our goal is to maximize it, subject to unitary vector $v^T v = 1$. Using Lagrange multipliers we have the optimization problem given in equation 1.

$$\begin{aligned} max \quad & v^T X X^T v \\ s.t. \quad & v^T v = 1 \end{aligned}$$

$$\begin{aligned} L(v) &= v^T X X^T v - \alpha(v^T v - 1) \\ \nabla L(v) &\overset{(a)}{=} \nabla v^T \Sigma v - \alpha \nabla v^T v \\ &\overset{(b)}{=} \nabla v^T (\Sigma v) + \nabla \left(v^T \Sigma\right) v - 2\alpha v^T \\ &= (\Sigma v)^T + v^T \Sigma - 2\alpha v^T \\ &= v^T \Sigma^T + v^T \Sigma - 2\alpha v^T \\ &= v^T \left(\Sigma^T + \Sigma\right) - 2\alpha v^T \\ 0 &\overset{(c)}{=} 2v^T \Sigma - 2\alpha v^T \\ v^T \Sigma &= \alpha v^T \end{aligned} \tag{1}$$

On statement (a) the matrix multiplication XX^T was called Σ (covariance matrix), on (b) gradient chain rule was applied, on (c) the Σ symmetry property was exploited and the set the expression equal to zero in order to find a critical point.

The critical point found led us to the relation $v_1^T \Sigma = \alpha v_1^T$, which is the well known eigen problem [55], [38], [24]. The solution for our problem is the first left singular vector $v = u_1$ of X and its associated singular value λ_1. By substitution of $v = u_1$ on the objective function of equation 1, we got the maximization function could be understood as maximizing the singular value λ_1 corresponding to the singular vector u_1.

$$\begin{aligned} v_1^T X X^T v_1 &= u_1^T \Sigma u_1 \\ &= u_1^T (\lambda u_1) \\ &= \lambda_1 \end{aligned} \tag{2}$$

We are maximizing a eigen value of the matrix $XX^T = \Sigma$, the solution is the eigen vector u_1 related to the largest eigenvalue λ_1. The projection $u_1^T X$ guarantees to have the maximum magnitude over Frobenius norm.

In addition is interesting to analyze the Hessian of $L(v)$, the resulting matrix tell information about the critical points found.

$$\begin{aligned} L(v) &= v^T X X^T v - \alpha v^T v \\ L(v) &= v^T (\Sigma - \alpha I) v \\ \nabla^2 L(v) &= \Sigma - \alpha I \end{aligned} \tag{3}$$

The matrix $\Sigma - \alpha I$ is singular, and its quadratic form could be positive or negative semidefinite. By construction the matrix $\Sigma = XX^T$ always have positive diagonal entries $\sigma_{ii} \geq 0, \forall i \in \{1, \ldots, n\}$. Matrix $\Sigma - \alpha I$ is negative semidefinite because $\sigma_{11} - \lambda_1 < 0$ due to the fact the largest eigenvalue is greater than any entry. The negative semidefinite quadratic function has a hill at zero through the direction of u_1 as depicted on figure 4.b. This hill means that $L(u)$ is maximum through u_1 direction.

The projection problem is a quadratic form, its plot with the unitary vector constraint is depicted on figure 4.a. The projection vector (eigen vector of Σ) u_1 is showed as a blue

vector pointing the maximum of the quadratic function. On figure 4.b is depicted the lagrange maximization function $L(v)$, you can notice the hill on the function and the blue eigen vector showing the maximum through that direction.

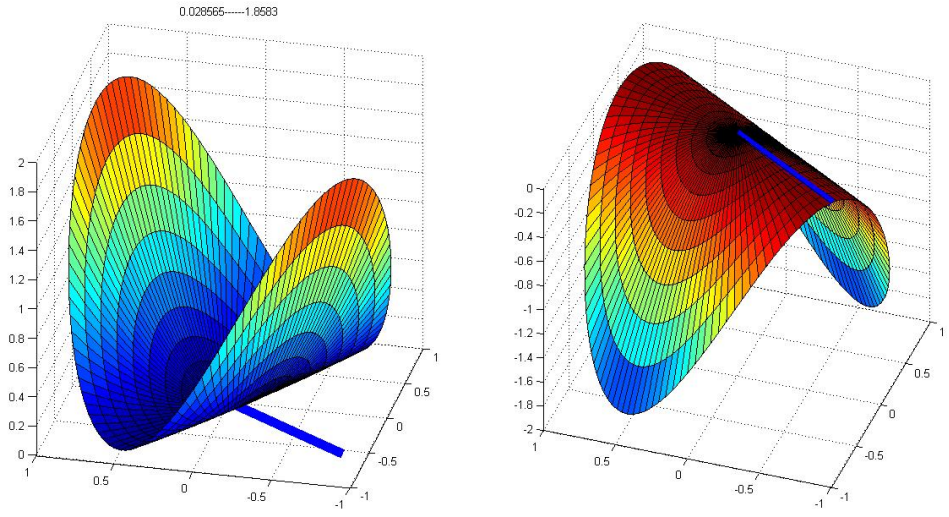


Figure 4: PCA quadratic form (a), Lagrange function (b)

The projection problem can be formulated in a similar way for higher dimensions, projecting on two dimensions at a time. Its important to recall from figure 4 that the principal components could be found rotated.

3.2 Independent Component Analysis

Consider the blind source separation problem where measurements of N sensors are available, during a time of length M. Each sensor measurement is assumed to be a linear combination of independent components. The goal is to find those independent sources [24], [25].

Each sensor $i \in \{1, \ldots, N\}$ has a measurement vector $x_i^T = [x_i(1), \ldots, x_i(M)]^T$ with length M. Lets consider each entry $x_i(j)$ of the measurement vector x_i^T, as a scalar random variable that distributes according to $x_i \sim P_{x_i}(x_i)$.

Each random variable $x_i(j)$ is a linear combination of independent sources $\{s_1(j), \ldots, s_N(j)\}$ according to equation 4.

$$x_i(j) = \sum_{n}^{N} a\,(i,n)\,s\,(n,j) = a_i^T s(j) \tag{4}$$

Graphically this linear combination for every source is expressed using matrix notation

on equation 5.

$$\begin{pmatrix} \boxed{x_i\{j\}} \end{pmatrix} = \begin{pmatrix} — & a_i^T & — \end{pmatrix} \begin{pmatrix} | \\ s(j) \\ | \end{pmatrix}$$
$$\begin{pmatrix} — & x_i^T & — \end{pmatrix} = \begin{pmatrix} — & a_i^T & — \end{pmatrix} \begin{pmatrix} — & s_1^T & — \\ & \vdots & \\ — & s_N^T & — \end{pmatrix} \tag{5}$$
$$X^T = AS^T$$

The probability density function of the random variable $x_i(j)$ in terms of the densities of the independent sources $\{s_1(j), \ldots, s_N(j)\}$ is given by the density of the linear combination $x_i(j) = a_i^T s(j)$. Consider each independent component distributes according $S_i \sim f_{S_i}(s_i)$ and the joint distribution is $S \sim \prod_i^n f_{S_i}(s_i)$.

From the appendix about function of a vector random variable we know that the joint distribution of vector X is the one on equation 6.

$$f_X(x) = f_S(A^{-1}x) \left|\det\left(J(A^{-1}x)\right)\right| \tag{6}$$

In order to estimate the independent components we applied maximum likelihood (ML) estimation. ML aims to find a set of parameters (matrix A) that maximize the likelihood function (joint distribution or probability of our observations matrix X). The likelihood is obtained as the product of the densities $f_X(x^{(j)})$ for the M vector samples $x^{(j)}$. Let's name A^{-1} as W, the likelihood on equation 7 is a function of W.

$$L(W) = \prod_{j=1}^{M} f_S(Wx^{(j)}) \left|\det(W)\right| \tag{7}$$

We used the natural logarithm on the likelihood to transform products to sums, maintaining the function maximum at the same point $\hat{W}$. The log likelihood is then given by

$$\log L(W) = \sum_{j=1}^{M} \log\left(f_s(Wx^{(j)})\right) + M \log\left|\det(W)\right| \tag{8}$$

In order to maximize the log likelihood a numerical maximization algorithm is needed. We used the gradient method. The first step is to take the gradient from the likelihood in order to compute the direction of maximum change.

$$\nabla_W log\left(L(W)\right) = \nabla_W \sum_{j=1}^{M} log\left(f_s(Wx^{(j)})\right) + \nabla_W M log\left|det(W)\right| \tag{9}$$

Let's assume a cdf for each independent source to be $F_{S_i}(s_i) = g(s_i) = 1/(1+e^{-s_i})$ and the pdf to be $g'(s_i)$. Applying gradient on the first term of equation 9 we got:

$$\begin{aligned}
\sum_{j=1}^{M} \nabla_W log\left(f_s(Wx^{(j)})\right) &\overset{(a)}{=} \sum_{j=1}^{M} \frac{\partial log\left(f_s(Wx^{(j)})\right)}{\partial f_s(Wx^{(j)})} \frac{\partial f_s(Wx^{(j)})}{\partial Wx^{(j)}} \frac{\partial Wx^{(i)}}{\partial W} \\
&\overset{(b)}{=} \sum_{j=1}^{M} \frac{1}{f_s(Wx^{(j)})} \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)}) \begin{pmatrix} 1-2g(w_1^T x^{(j)}) \\ \vdots \\ 1-2g(w_n^T x^{(j)}) \end{pmatrix} x^{(j)T} \\
&\overset{(c)}{=} \sum_{j=1}^{M} \begin{pmatrix} 1-2g(w_1^T x^{(j)}) \\ \vdots \\ 1-2g(w_n^T x^{(j)}) \end{pmatrix} x^{(j)T}
\end{aligned} \tag{10}$$

The gradient of the first term of the likelihood was depicted on equation 10, where we applied the chain rule for gradient for step (a). On (b) we applied the derivative of a logarithm, gradient with respect to a vector $Wx^{(i)}$ according to equation 11 and gradient of a vector with respect to a matrix W.

$$\begin{aligned}
\frac{\partial f_s(Wx^{(j)})}{\partial Wx^{(j)}} &\overset{(a)}{=} \frac{\partial \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)})}{\partial Wx^{(j)}} \\
&\overset{(b)}{=} \begin{pmatrix} \frac{\partial \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)})}{\partial w_1^T x^{(j)}} \\ \vdots \\ \frac{\partial \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)})}{\partial w_n^T x^{(j)}} \end{pmatrix} \\
&= \begin{pmatrix} \frac{f'_{s_1}(w_1^T x^{(j)}) \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)})}{f_{s_1}(w_1^T x^{(j)})} \\ \vdots \\ \frac{f'_{s_n}(w_1^T x^{(j)}) \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)})}{f_{s_n}(w_1^T x^{(j)})} \end{pmatrix} \\
&\overset{(c)}{=} \prod_{i=1}^{n} f_{s_i}(w_i^T x^{(j)}) \begin{pmatrix} \frac{g'_1(1-2g_1)}{g'_1} \\ \vdots \\ \frac{g'_n(1-2g_n)}{g'_n} \end{pmatrix}
\end{aligned} \tag{11}$$

On (a) we applied the definition of joint distribution of independet variables. On (b) we applied the definition of gradient of a scalar with respect to a vector. On (c) we used the definition of derivative of sigmoid, $g'' = g'(1-2g)$.

The gradient for the second term on equation 9 is the well known gradient of a determinant.

$$\nabla_W M log\left|det\left(W\right)\right| = MW^{-T} \tag{12}$$

Replacing 10 and 12 on 9 we got.

$$\nabla_W log\left(L(W)\right) = \sum_{j=1}^{M} \begin{pmatrix} 1-2g(w_1^T x^{(j)}) \\ \vdots \\ 1-2g(w_n^T x^{(j)}) \end{pmatrix} x^{(j)T} + MW^{-T} \tag{13}$$

The gradient led us to the updating rule

$$W_{new} = W_{old} + \alpha \sum_{j=1}^{M} \begin{pmatrix} 1-2g(w_1^T x^{(j)}) \\ \vdots \\ 1-2g(w_n^T x^{(j)}) \end{pmatrix} x^{(j)T} + MW^{-T} \tag{14}$$

A stochastic version of this rule could be used omitting the expectation $\sum_{j=1}^{M}$ thus using a single sample $x^{(j)}$ on each step.

$$W_{new} = W_{old} + \alpha \begin{pmatrix} 1 - 2g(w_1^T x^{(j)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(j)}) \end{pmatrix} x^{(j)T} + W^{-T} \tag{15}$$

This algorithm provides an estimation of the mixing matrix $W = A^{-1}$. Once it converges, the independent sources are computed as $S = WX$.

It was assumed the entries of a sensor measurement vector $x_i = \{x_i(1), \ldots, x_i(M)\}$ were independent and identically distributed. This assumption let write the joint distribution $P(x_i) = \prod_j^M P(x_i(j))$. Such an assumption could be counter intuitive because of the correlation between samples e.g. audio sources. Nevertheless this approach showed good performance and there is the option to shuffle samples as a pre-process step.

3.2.1 ICA Simulations

In this section the objective was to illustrate the computation of independent components from simulated data. Each sample (i) of the independent sources $s^{(i)} = [s_1^{(i)}, s_2^{(i)}]^T$ was extracted from two independent uniform distributions on the $[-0.5, 0.5]$ interval, i.e. $s \sim U(-0.5, 0.5)$. Samples were plotted on figure 5.a, it can be seen them are uniformly distributed over an square.

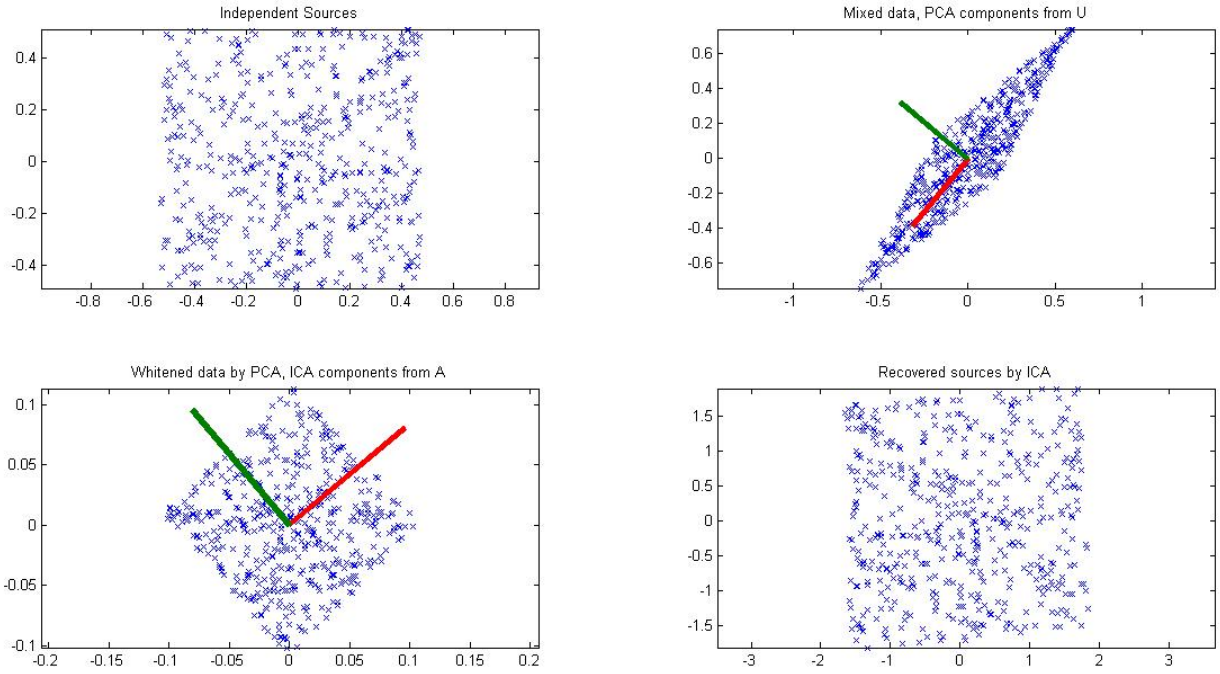


Figure 5: Linear transformations to recover independent sources by ICA

Mixed components x were computed by means of a random mixing matrix A, i.e. $x = As$. The mixed samples were plotted on figure 5.b having a parallelogram structure.

The next step to recover the independent components was to apply whitening as preprocessing.
Whitening of a random vector is an orthogonal linear transformation $z = Fx$ such that the components of the resulting vector z are uncorrelated and scaled to unitary variance. The resulting covariance matrix is the identity $E(zz^T) = I$.

There are several ways to whiten data, one of them is by the eigen value decomposition of the covariance matrix $E(xx^T) = U\Sigma U^T$. The orthogonal matrix U which contains the eigen vectors of $E(xx^T)$ uncorrelate the variables and the inverse of the eigen values matrix $\Sigma^{-1/2}$ scale the variance to the unity. Lets compute the covariance matrix of the transformed variable $z = Fx$, using $F = \Sigma^{-1/2}U^T$.

$$\begin{aligned} E(zz^T) &= E(Fxx^TF^T) = FE(xx^T)F^T \\ &= FU\Sigma U^TF^T \\ &= \Sigma^{-1/2}U^TU\Sigma U^TU\Sigma^{-1/2} \\ &= \Sigma^{-1/2}\Sigma\Sigma^{-1/2} \\ &= I \end{aligned} \tag{16}$$

The whitened data was plotted on figure .c, where its noted the square shape was recovered from the parallelogram on figure 3.b. The eigen vectors from U were plotted on figure 5.b as a red and green line.

The final step to recover the independent sources was to apply ICA on the whitened data using the algorithm described on the previous section. Its worth to note that the substitution of the mixing transformation $x = As$ on the identity covariance matrix $E(zz^T) = I$ we get $\tilde{A} = FA$ orthogonal transformation.

$$\begin{aligned} E(zz^T) = I &= E(Fxx^TF^T) \\ &= (FA)E(ss^T)(A^TF^T) \\ &= (FA)I(A^TF^T) \\ &= \tilde{A}\tilde{A}^T = I \end{aligned} \tag{17}$$

This derivation tell us that the mixing matrix A is a factor of the orthogonal transformation $\tilde{A} = FA$. The fact that $\tilde{A}$ is orthogonal restrict the search for the mixing matrix to the space of orthogonal matrices. Instead of estimating n^2 parameters, an orthogonal matrix only have $n(n-1)/2$ degrees of freedom. This advantage increase the performance of the numerical estimation algorithm in terms of computational time. The estimated sources are plotted on figure 5.d and the projection vectors (columns of A) are plotted as a red and a green line on figure 5.c. Another example for ICA is presented on figure 6 using as independent sources a random variable with uniform distribution and an exponential.

Whitening give Independent Components only up to an orthogonal transformation. This means data could be rotated and also be uncorrelated with unit variance. For example consider the whitening matrix $F = \Sigma^{-1/2}U^T$ multiplied on the left by the orthogonal matrix U. This orthogonal transformation again to un correlated variables with identity covariance matrix. Lets name the new whitening matrix $\tilde{F} = U\Sigma^{-1/2}U^T$, $z = \tilde{F}x$

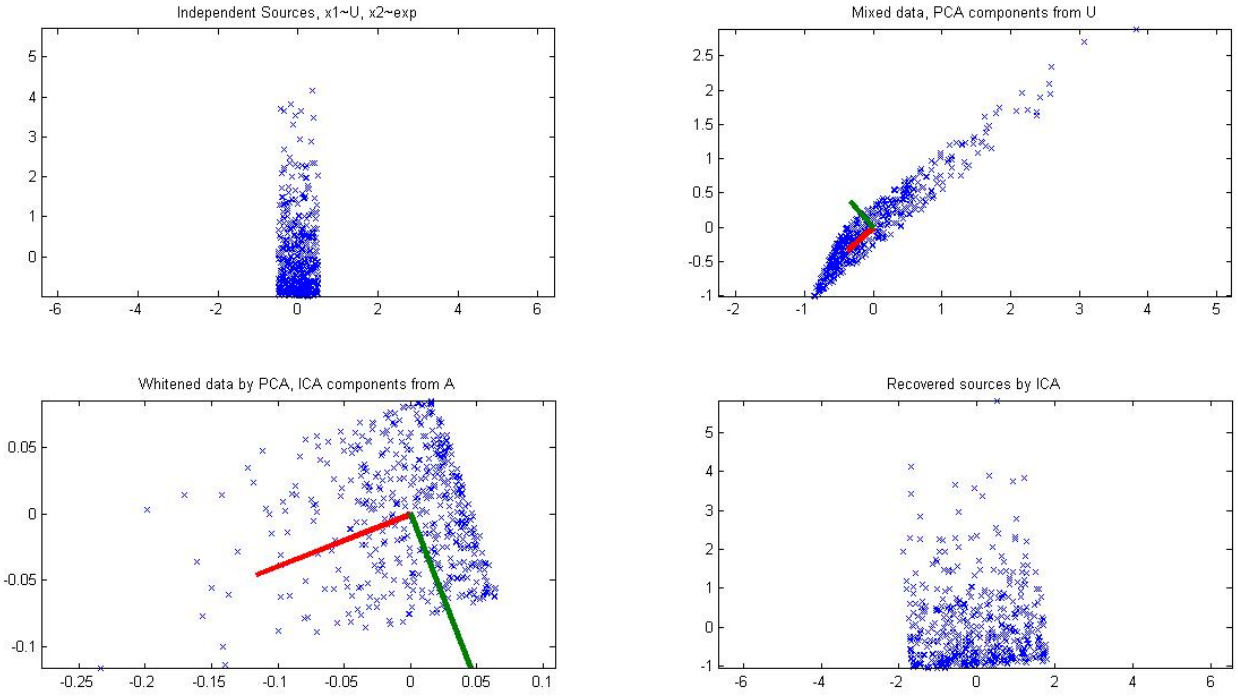


Figure 6: Recover uniform and exponential sources by ICA

$$
\begin{aligned}
E(zz^T) &= E(\tilde{F}xx^T\tilde{F}^T) = \tilde{F}E(xx^T)\tilde{F}^T \\
&= \tilde{F}U\Sigma U^T\tilde{F}^T \\
&= U\Sigma^{-1/2}U^TU\Sigma U^TU\Sigma^{-1/2}U^T \\
&= U\Sigma^{-1/2}\Sigma\Sigma^{-1/2}U \\
&= UIU^T = I
\end{aligned} \tag{18}
$$

This behavior is represented on figure 7, where the orthogonal transformation of the samples is represented by a rotation. The transformation led to the identify matrix as showed on the right column of figure 7.

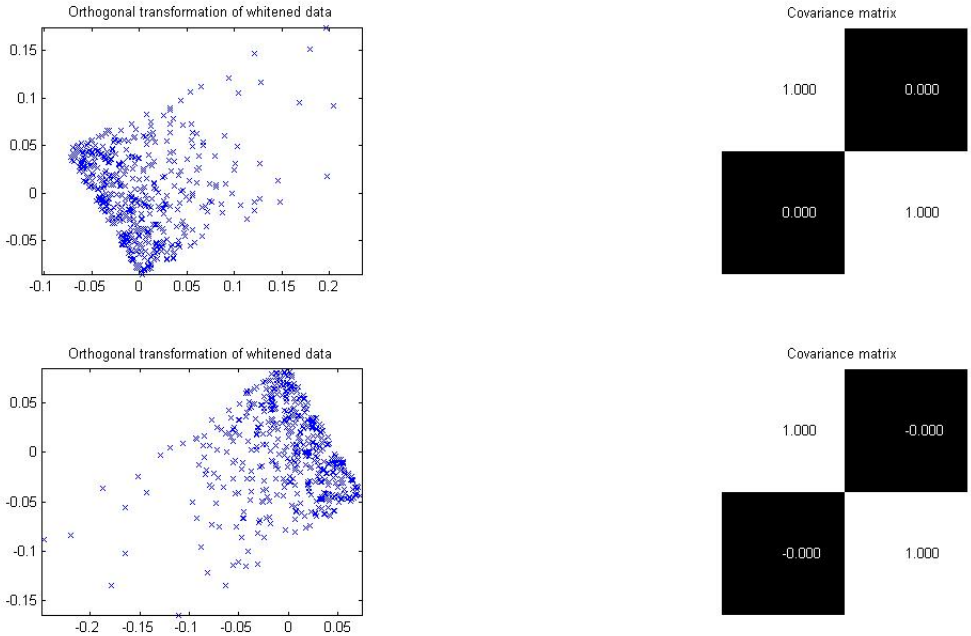


Figure 7: Orthogonal transformation of whitened data

3.3 Partial Least Squares

This section discuss the Partial Least Squares also known as Projection to Latent Space method for data representation.

3.3.1 PLS Model

Suppose two data sets available: X the predictors (regressors, independent variables) and Y the responses (regressions, dependent variables). The task is to find a representation for both data sets such that maximizes their correlation. The dataset contained in the matrix X was arranged according graphical equation 19, where the rows represents variables or dimensions $\{x_1, \cdots, x_n\}$ and columns represent different samples $\{x^1, \cdots, x^{(m)}\}$.

$$variables \underbrace{\left\{ \begin{pmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{pmatrix} \right.}_{samples} = X \tag{19}$$

Consider the singular value decomposition (SVD) of matrix $X = U\Sigma V^T$. A well known representation for dimensionality reduction of matrix X is $\hat{X} = U_r^T X = \Sigma V^T$. Where U_r^T is the reduced singular vector matrix U to r column vectors. This representation use the magnitude of each sample projection into the U_i direction. Additionally the representation $\hat{X} = U_r^T X$ has uncorrelated components.
The representation given by SVD, $X = U(\Sigma V^T)$ can be thought as two factors: the first one is U which represents direction and the second (ΣV^T) represents projections magnitudes.
The PLS literature work with the transpose of data matrix $X^T = (V\Sigma)U^T = TP^T$,

and define the score (projections) matrix $T = (V\Sigma)$ and the loading (directions) matrix $P = U$.

The PLS method seeks for a representation for both sets, the regressors X and regression Y. SVD perform a representation for matrix $X = TP^T$, PLS look after a transformation W such that $T = W^T X$ have components maximally correlated with the transformed components of $Q = Y^T C$.

As illustration consider the inner product between the samples of x_i and the regression samples y_j. This product is related to the correlation between variables x_i and y_i. The goal is to find transformations W and C such that maximize the product of the transformed variables t_i and q_j. The product square $w_i^T XY^T c_j = t_i q_j^T$ is the function to be maximized. Problem statement on equation 20.

$$\begin{aligned} & max \quad (w_i^T XY^T c_j)^2 \\ & s.t. \\ & w^T w = 1 \\ & c^T c = 1 \end{aligned} \tag{20}$$

Using Lagrange multipliers the function to be maximized was stated on equation 21. Then gradient with respect to vectors w and c was applied and equal to zero in order to find critical points.

$$\begin{aligned} L(w,c) &= (w^T XY^T c)^2 - \alpha w^T w - \beta c^T c \\ \nabla_w L(w,c) &= 2w^T XY^T c(XY^T c)^T - 2\alpha w^T \\ 0 &= w^T XY^T c(XY^T c)^T - \alpha w^T \\ 0 &= w^T \left(XY^T c(XY^T c)^T - \alpha I \right) \end{aligned} \tag{21}$$

Gradient equal to zero on equation 21 depicted the eigen problem for rank one symmetric matrix $XY^T c(XY^T c)^T$. This matrix has only one eigenvalue different of zero and its eigenvector is $w^T = (XY^T c)^T$. The eigenvalue is computed as $\alpha = (XY^T c)^T XY^T c$.

Gradient with respect to c was computed on equation 22.

$$\begin{aligned} \nabla_c L(w,c) &= 2w^T XY^T c(w^T XY^T) - 2\beta c^T \\ 0 &= w^T XY^T c(w^T XY^T) - \beta c^T \end{aligned} \tag{22}$$

Lets replace the vector $w^T = (XY^T c)^T$ on equation 22.

$$\begin{aligned} 0 &= (XY^T c)^T XY^T c((XY^T c)^T XY^T) - \beta c^T \\ 0 &= c^T YX^T XY^T cc^T YX^T XY^T - \beta c^T \\ 0 &= c^T (YX^T XY^T cc^T YX^T XY^T - \beta I) \\ 0 &= c^T \left((YX^T XY^T c)(YX^T XY^T c)^T - \beta I \right) \end{aligned} \tag{23}$$

Equation 23 depicted the eigen value problem for rank one symmetric matrix $(YX^T XY^T c)(YX^T XY^T$ This relation lead us to the eigen value problem of the symmetric matrix $YX^T(YX^T)^T$

$$\beta c = YX^T XY^T c \tag{24}$$

Where c is the eigen vector of $YX^T(YX^T)^T$. The same derivation follow for vector w, which is the eigen vector for symmetric matrix $XY^T(XY^T)^T$.

$$\alpha w^T = w^T XY^T(XY^T)^T \tag{25}$$

Results on equations 24 and 25 refer to the right and left singular vectors of matrix XY^T. The constants α and β refer to the corresponding singular values.

In order to tell the shape of $L(w, x)$ lets examine its Hessian $\nabla^2_w L(w, x)$ and $\nabla^2_c L(w, x)$. The goal is to test for a maximum, minimum, saddle point or singular case.

$$\begin{aligned}
\nabla^2_w L(w, x) &= \nabla^2_w (w^T(XY^T)c)^2 - \alpha\nabla^2_w w^T w - \beta\nabla^2_w c^T c \\
&= \nabla_w 2w^T(XY^T c)(XY^T c)^T - \alpha\nabla_w 2w^T \\
\nabla^2_w L(w, x) &= (XY^T c)(XY^T c)^T - \alpha I_{nx}
\end{aligned} \tag{26}$$

The Hessian with respect to c was computed on equation 27.

$$\nabla^2_c L(w, x) = (w^T XY^T)(w^T XY^T)^T - \beta I_{ny} \tag{27}$$

Where nx and ny are the number of predictor and prediction variables. The results for both Hessians led singular matrices $(XY^T c)(XY^T c)^T - \alpha I_{nx}$ and $(w^T XY^T)(w^T XY^T)^T - \beta I_{ny}$ which are negative definite. This implies that $L(w, c)$ has maximum at the eigen vectors w and c. The negative definiteness could be explained similar to the used on PCA section.

The question that naturally arises is: Why this representation is useful? Its useful because the maximally correlated transformed variables could be used for regression and the transformed variables with low correlation could be discarded. The transformed variables with low correlation could be understood as part of the data not related to the regression, possibly noise.

3.3.2 PLS simulation

The vector w_i computed on equation 25 transforms the variables of matrix X into a variable z_i which has the property of maximally correlation with the regression variable Y.

As example consider the PLS transformation of two regressors variables x_1 and x_2 which are samples or two independent uniform distributions. The regression variable y is the sum of the regressors variables, for simplicity data follows a linear model, i.e. $y = x_1 + x_2$.

On figure 8 a simulation was performed in order to find the correlated variables z. On figure 8.a the original set up is showed by red points the regressors and blue the regression. On figure 8.b and 8.c the relation between x_i and y is plotted. It can be seen that a linear regression is difficult for each variable x_i. On figure 8.d the transformed regressors z_i were plotted, it's noted that the transformation correspond to a rotation. Finally on figure 8.e and 8.f the transformed variable were compared to the regression. Figure 8.e reflected a quite notorious linear behavior between z_1 and y. Figure 8.f reflected no correlation between z_2 and y.

In fact the correlation matrix for $[Z|Y]$ showed high correlation between z_1 and y and poor correlation for z_2 and y. Further more the SVD used to compute w and c could be used to tell how many transformed variables to pick. In this example the singular value related to z_1 was greater than z_2 and z_2 was near to zero, suggesting to use only z_1 for regression and reducing the dimensionality.

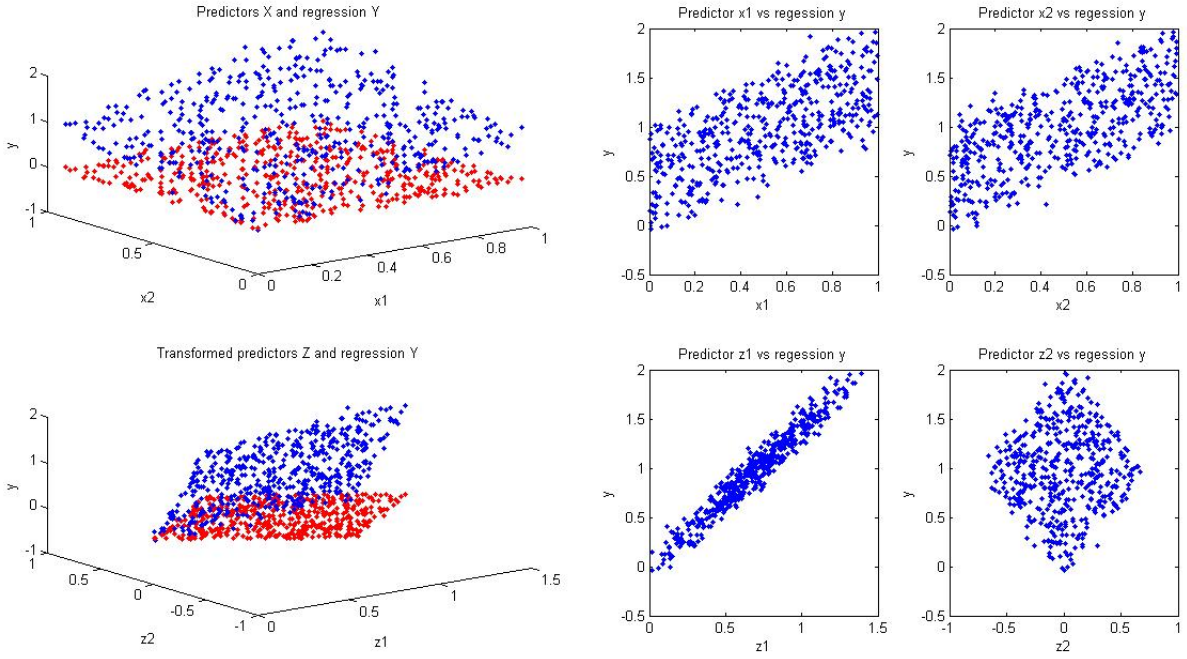


Figure 8: Partial Least Squares transformation

4 Multiway data analysis

4.1 Notation and Definitions

Nth-order tensors (multiway arrays) are denoted by underlined capital letters, matrices (two-way arrays) by capital letters, and vectors by lower case letters. The ith entry of a vector x is denoted by x_i, element (i, j) of a matrix X is denoted by X_{ij}, and element $(i_1, i_2, \cdots, i_N)$ of an Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ by $\underline{X}_{i_1, i_2, \cdots, i_N}$. Indexes typically range from 1 to their capital version, for example, $i_N \in \{1, \cdots; , I_N\}$.

A fiber on mode n was obtained by fixing all indexes but the *nth* index, e.g. the fibers on mode 1 of tensor $\underline{X}$ are the columns $\underline{X}(:, i_2, \ldots, i_N)$. The fibers on each mode were depicted on figure 9.

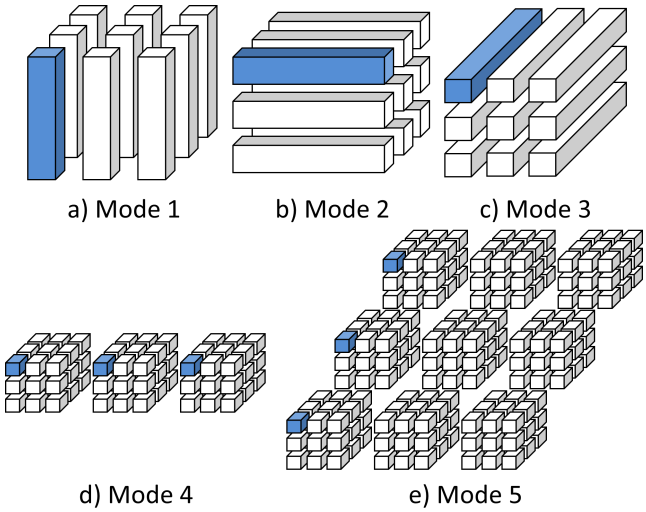


Figure 9: Tensor fibers for different modes and orders. On a),b),c) a third order tensor was depicted, on d) a fourth order and on e) an fifth order tensor. Mode 1 fibers are columns, mode 2 are rows, and mode 3 fibers go across the tensor. Mode 4 fibers, pick a single element from each third order tensor moving across the fourth index. Mode 5 fibers, pick a single element from each third order tensor moving across the fifth index.

The mode-n unfolding of a tensor is denoted by $X^{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$. An unfolding was build by arranging n-mode fibers as columns of the unfolding matrix $X^{(n)}$. On figure 10 mode 1,2 and 3 unfoldings of a third order tensor were presented.

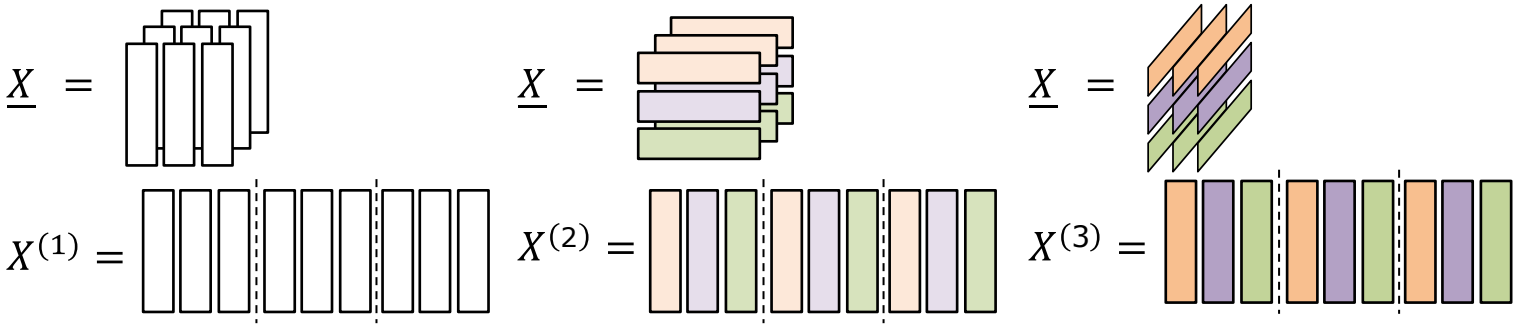


Figure 10: Unfolding of tensor $\underline{X}$ on mode 1, 2 and 3, $X^{(1)}$, $X^{(2)}$, $X^{(3)}$ respectively.

The n-mode product between a tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and a matrix $A \in \mathbb{R}^{J_n \times I_n}$ is denoted by $\underline{Y} = \underline{X} \times_n A \in \mathbb{R}^{I_1 \times \cdots \times J_n \times \cdots \times I_N}$. The product on mode n requires the number of columns on matrix A be equal to the number of elements on the n-mode fiber from $\underline{X}$.

4.1.1 Tucker decomposition

Consider a real Nth order tensor $\underline{X}$ of size $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. Tucker decomposition (TD) factorize a tensor into a core tensor and a set of factor matrices $U^{(n)}$ according to 79.

$$\begin{aligned} \underline{X} &= \underline{G} \times_1 U^{(1)} \times_2 \cdots \times_N U^{(N)} \\ &= [\![G, U^{(1)}, \cdots, U^{(N)}]\!] \end{aligned} \tag{28}$$

The *nth* factor matrix $U^{(n)}$ was build by the left singular vectors of the tensor unfolded on mode n, $X^{(n)} \in \mathbb{R}^{I_n \times I_1 \ldots I_{n-1} I_{n+1} \ldots I_N}$. The singular value decomposition of the mode n unfolding has the form of equation 80 .

$$X^{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)T} \tag{29}$$

The matrix of left singular vectors of the *nth* unfolding is the factor matrix $U^{(n)}$. This factor matrices $U^{(n)}$ are orthogonal and its transpose is used to compute the core tensor $\underline{G}$ by equation 84.

$$\underline{X} \times_1 U^{(1)T} \times_2 \cdots \times_N U^{(N)T} = \underline{G} \tag{30}$$

Tucker decomposition can be computed according to algorithm 1.

Algoritmo 1: Tucker decomposition

Input : Nth order tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
Output: Core tensor $\underline{G}$ and factor matrices $\{U^{(1)}, \ldots, U^{(N)}\}$
for *each unfolding* $\underline{X}^{(n)}$, $n \in \{1, \ldots, N\}$ **do**
  $X^{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)T}$
Compute core tensor $\underline{G} = \underline{X} \times_1 U^{(1)T} \times_2 \cdots \times_N U^{(N)T}$

The *nth* factor matrix $U^{(n)}$ was composed by singular vectors which formed a basis for the fibers on mode n. A subspace approximation for the tensor could be performed by truncating the factor matrices up to r singular vector on each factor matrix. The truncated factor matrix was denoted by $\hat{U}_r^{(n)}$. Truncation affected the core tensor $\hat{\underline{G}}$, which also have to be truncated. Tensor approximation was computed using equation 79.

$$\begin{aligned} \hat{\underline{X}} &= \hat{\underline{G}} \times_1 \hat{U}_r^{(1)} \times_2 \cdots \times_N \hat{U}_r^{(N)} \\ &= [\![\underline{G}; \hat{U}_r^{(1)}, \cdots, \hat{U}_r^{(N)}]\!] \end{aligned} \tag{31}$$

4.1.2 Subspace approximation.

The subspace approximation of an *Nth* order tensor $\underline{X} \in \mathbb{R}^{I_1, \ldots, I_N}$ by a multilineal rank-$(J_1, \ldots, J_N)$ comes from the tensor HOSVD according to equation 79. In order to perform a subspace approximation, singular vectors from the factor matrices have to be removed. The multilineal approximation on the *nth* mode eliminates columns of the *nth* factor matrices $U^{(n)}$ and eliminates *nth*-frames from the core tensor $\underline{G}$. The tensor approximation has the same form as the tucker approximation.

$$\hat{\underline{X}} = \hat{\underline{G}} \times_1 \hat{U}^{(1)}_{J_1} \times_2 \cdots \times_N \hat{U}^{(N)}_{J_N} \tag{32}$$

Where core tensor was truncated as $\hat{\underline{G}} \in \mathbb{R}^{J_1,\ldots,J_N}$, $J_n \leq I_n$, and factor matrices $\hat{U}^{(n)}_{J_n} \in \mathbb{R}^{I_n \times J_n}$.

An example of the tensor subspace approximation was presented graphically on figure 11, using a third order tensor $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and a multilineal rank-$(1, I_2, I_3)$ approximation (block Tucker approximation [14]).

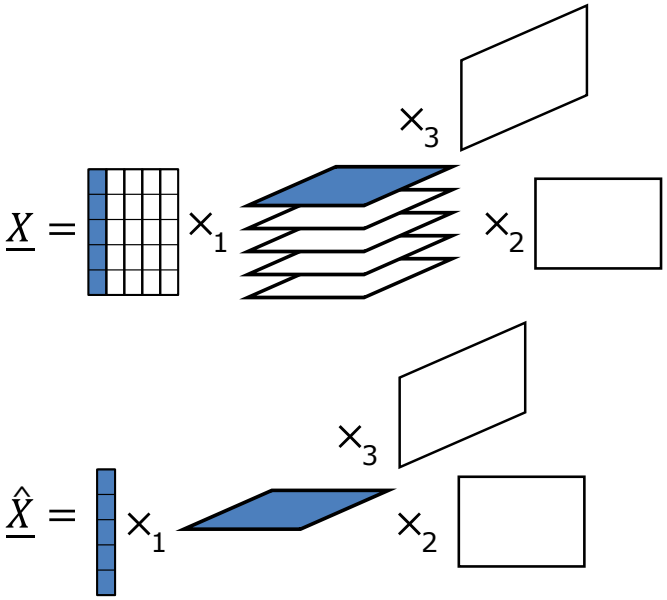


Figure 11: Tensor subspace approximation of $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, by a multilineal rank-$(1, I_2, I_3)$ block Tucker approximation.

4.2 Parallel factor analysis

The tensor decomposition PARAFAC (Parallel factor analysis) also known as CANDECOMP (Canonical decomposition) or CP (Canonical Polyadic), define a N order tensor $\underline{X}$ as a sum of rank-one tensors according equation ...

$$\underline{X} = \sum_{r=1}^{R} \lambda_r a^{(1)}_r \circ a^{(2)}_r \circ \cdots \circ a^{(N)}_r \tag{33}$$

Where each rank one tensor is composed by outer products of vectors $a_r(n)$ (Columns of factor matrices $A^{(n)}$) scaled by a scalar factor λ_r. PARAFAC model can be represented by 79 when the core tensor is super diagonal.

4.2.1 Inner Product and Covariance tensor

The *inner product* of two tensors (same order and dimensions) $\underline{X}, \underline{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is defined as the sum of the product between elements with the same positions (equation 34). The squared Frobenius norm is $||\underline{X}||^2_F = \langle \underline{X}, \underline{X} \rangle$.

$$\langle \underline{X}, \underline{Y} \rangle = \sum_{i_1 \cdots i_N} \underline{X}_{i_1 \cdots i_N} \underline{Y}_{i_1 \cdots i_N} \tag{34}$$

The *n-mode cross covariance* between an Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and an Mth-order tensor $\underline{Y} \in \mathbb{R}^{J_1 \times \cdots \times I_n \times \cdots \times J_N}$ with the same size I_n on the nth mode is defined as equation 77.

$$\begin{aligned} &\underline{C} = COV_{\{n;n\}}\left(\underline{X}, \underline{Y}\right) = \langle \underline{X}, \underline{Y} \rangle_{\{n;n\}} \\ &COV_{\{n;n\}}\left(\underline{X}, \underline{Y}\right) \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{n-1} \times J_{n+1} \times \cdots \times J_M} \end{aligned} \tag{35}$$

The symbol $\langle \bullet, \bullet \rangle$ represents an *n-mode multiplication* between two tensors. Its element-wise computation was defined on equation 78 and its complete computation using outer product was depicted on equation 37.

$$\begin{aligned} &c_{i_1, \cdots, i_{n-1}, i_{n+1}, \cdots, i_N, j_1, \cdots, , j_{n-1}, j_{n+1}, \cdots, j_M} \\ &= \sum_{i_n=1}^{I_n} x_{i_1, \cdots, i_n, \cdots, i_N} y_{j_1, \cdots, i_n, \cdots, j_M} \end{aligned} \tag{36}$$

$$C = \sum_{i_n=1}^{I_n} \sum_{r=1}^{R} \lambda_r X^{(i)} \circ u_r \circ v_r \tag{37}$$

The computation of the covariance tensor (CT) is similar to the computation of sample covariance matrix $\Sigma = XX^T$, where columns $x^{(i)}$ of matrix X are samples. In that case the covariance matrix is computed as the sum of outer product $x^{(i)} \circ x^{(i)T}$ for each sample vector, as depicted on figure 22.

For the covariance tensor, samples $X^{(i)}$ are multilinear arrays with order $N \geq 2$. In that case the covariance tensor is computed as the sum of outer product $X^{(i)} \circ X^{(i)T}$ for each sample, as depicted on figure 22.

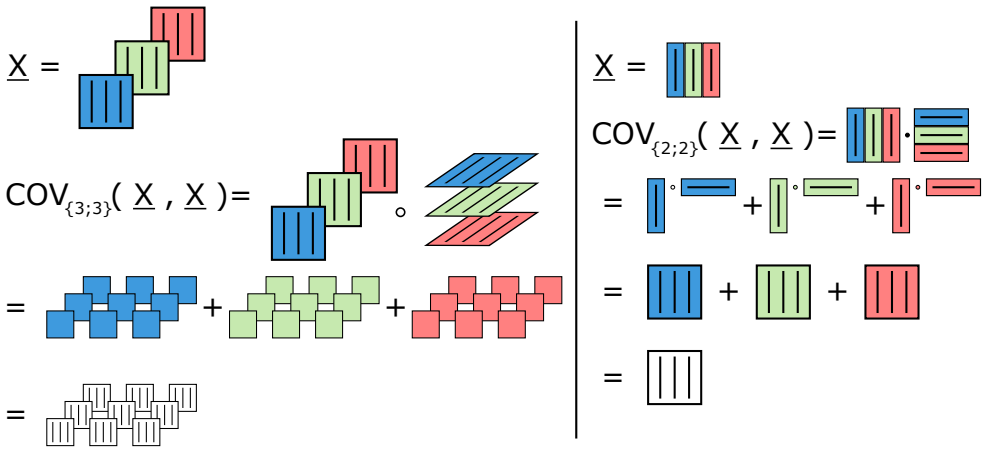


Figure 12: Tensor covariance for $\underline{X} \in \mathbb{R}^{3 \times 3 \times 3}$. Samples are frontal frames (mode-3 frame) $X^{(i)} \in \mathbb{R}^{3 \times 3}$. Resulting covariance tensor has order four.

A special case for the cross covariance tensor between a multiway array $\underline{X}$ and a matrix $Y \in \mathbb{R}^{J_1 \times I_n}$ results in a mode-n product.

$$COV_{\{n;2\}}\left(\underline{X}, Y\right) = \underline{X} \times_n Y. \tag{38}$$

For the computation of covariance tensor the n-mode column of $\underline{X}$ and columns of Y should be centered.

4.3 Application of Tensor Decomposition

Numerous applications of multiway data analysis have been proposed along different disciplines. This section presents an interesting application where tensor decomposition is applied on the estimated fourth order statistics from n mixed sources. The estimated statistics build up according to a fourth order tensor. The goal is to find a linear transformation that makes the fourth order joint cumulants zero (find independent components).

4.3.1 Higher Order Statistics

First characteristic function. The first characteristic function $\Phi_X(\omega)$ of a random variable X is defined as the expected value of $e^{j\omega X}$, with $\omega \in \mathbb{R}$.

$$\Phi_X(\omega) = E\{e^{j\omega X}\} = \int_{-\infty}^{\infty} p_X(x) e^{j\omega x} dx \tag{39}$$

Second characteristic function. The second characteristic function $\Psi_X(\omega)$ of a random variable X is defined as the Neperian logarithm of the first characteristic function.

$$\Psi_X(\omega) = ln\Phi_X(\omega) \tag{40}$$

Moments. The Nth order moment m_N^X of a random variable X is defined as the expected value of the Nth power of X.

$$m_N^X = E\{X^N\} \tag{41}$$

The Nth order moment of X can be obtained from the ω^N coefficient from the Taylor series expansion of the first characteristic (generating) function.

$$\begin{aligned} \Phi_X(\omega) = E\{e^{j\omega X}\} = \int_{-\infty}^{\infty} e^{j\omega x} f_X(x) dx = \int_{-\infty}^{\infty} \left(\sum_{k=0}^{\infty} \frac{x^k (j\omega)^k}{k!} \right) f_X(x) dx \\ = \sum_{k=0}^{\infty} \int_{-\infty}^{\infty} \left(\frac{(j\omega)^k}{k!} \right) \left(x^k f_X(x) \right) dx = \sum_{k=0}^{\infty} \frac{(j\omega)^k}{k!} E(x^k) \end{aligned} \tag{42}$$

Additionally the Nth order moment of X can be obtained differentiating the moment generating function N times and evaluating at zero.

$$m_N^X = \frac{1}{j^n} \frac{d^N \Phi_x(\omega)}{d\omega^N} \Big|_{\omega=0} \tag{43}$$

Multivariate Moment. The joint moment for a set of random variables $\{X1, \cdots, X_N\}$ is defined as the expected valued of their product:

$$Mom\,(X1, \cdots, X_N) = E\{X1 X_2 \cdots X_N\} \tag{44}$$

The second order joint moment of two different random variables is known as cross correlation.

Nth order moment of random vector. The Nth order moment of a random vector X is an Nth order tensor according to:

$$\mathcal{M}_N^X = E\{X \circ X \cdots \circ X\} \tag{45}$$

Cumulant The Nth order cumulant c_N^X of a random variable X could be obtained from the Taylor series expansion of the second characteristic (generating) function.

$$c_N^X = \frac{1}{j^n} \frac{d^N \Psi_x(\omega)}{d\omega^N}\Big|_{\omega=0} \tag{46}$$

Multivariate Cumulant The joint cumulant for a set of random variables $\{X_1, \ldots, X_N\}$ with zero mean can be defined in terms of moments according to:

$$\begin{aligned} Cum(X_1) &= E\{X_1\} \\ Cum(X_1, X_2) &= Mom(X_1, X_2) = E\{X_1 X_2\} \\ Cum(X_1, X_2, X_3) &= E\{X_1 X_2 X_3\} \\ Cum(X_1, X_2, X_3, X_4) &= E\{X_1 X_2 X_3 X_4\} - E\{X_1 X_2\} E\{X_3 X_4\} \\ &\quad - E\{X_1 X_3\} E\{X_2 X_4\} - E\{X_1 X_4\} E\{X_2 X_3\} \end{aligned} \tag{47}$$

It is important to state that the moments and cumulants for a scalar random variable (univariate case) is a scalar. the moments and cumulants for a scalar random variable (univariate) is a scalar. The moments and cumulants for a random vector (multivariate case) are arranged as tensors.

Sum of cumulants. Cumulants of a sum are the sum of the cumulants.

$$cum(X_1 + Y_1, X_2, \ldots, X_N) = cum(X_1, X_2, \ldots, X_N) + cum(Y_1, X_2, \ldots, X_N) \tag{48}$$

Scaling of cumulant. The joint cumulant of random variables $X_1, X_2, \ldots, X_N$ multiplied by constants $a_1, a_2, \ldots, a_N$ is the multiplication of the constants by the joint cumulant of $X_1, X_2, \ldots, X_N$.

$$cum(a_1 X_1, a_2 X_2, \ldots, a_N X_N) = \left(\prod_{i=1}^{N} a_i \right) cum(X_1, X_2, \ldots, X_N) \tag{49}$$

$$cum(X_1 + Y_1, X_2, \ldots, X_N) = cum(X_1, X_2, \ldots, X_N) + cum(Y_1, X_2, \ldots, X_N) \tag{50}$$

4.3.2 ICA by tensorial method

Consider the case where data follows the ICA model. Whitening the data we have:

$$z = VAs = W^T s \tag{51}$$

Where vector $s \in \mathbb{R}^n$ represented n independent sources, vector z represents the whitened observed mixtures, A is the mixing matrix, V is the whitening matrix and W^T is an orthonormal matrix. W is the separating matrix.

Let the fourth order cumulant tensor of z be denoted by $\mathcal{C}_4^Z = F$. According to [], an estimation of the Nth moment $\hat{m}_N^Z$ and the fourth order cumulant $\hat{F}$ using T independent samples $z^{(i)}$, could be computed using:

$$\hat{m}_N^Z = \frac{1}{T} \sum_{t=1}^{T} z_t^N \tag{52}$$

$$\hat{F} = \hat{c}_4^Z = \frac{T^2}{(T-1)(T-2)(T-3)} \left[(T+1)\hat{m}_4^Z - 3(T-1)(\hat{m}_2^Z)^2 \right] \tag{53}$$

Any entry $F_{i,j,k,l}$ of the cumulant tensor is given by the cross cumulant according to:

$$F_{i,j,k,l} = cum(z_i, z_j, z_k, z_l) \tag{54}$$

The cumulant tensor is a symmetric linear operator, thus have an eigenvalue decomposition. Let M be a eigen matrix of the tensor F. The eigenvalue decomposition is denoted by:

$$F(M) = \lambda M \tag{55}$$

Let define the eigen matrix as $M = w_m w_m^T$. The $\{i, j\}$ element of the product $F(M)$ is computed as:

$$\begin{aligned}
F_{i,j}(M) &= \sum_{k=1}^{n}\sum_{l=1}^{n} M_{k,l} cum(z_i, z_j, z_k, z_l) \\
F_{i,j}(w_m w_m^T) &= \sum_{k=1}^{n}\sum_{l=1}^{n} w_{mk} w_{ml} cum(z_i, z_j, z_k, z_l) \\
&\overset{(a)}{=} \sum_{k=1}^{n}\sum_{l=1}^{n} w_{mk} w_{ml} cum(\sum_{q=1}^{n} w_{qi} s_q, \sum_{q'=1}^{n} w_{q'j} s_{q'}, \sum_{r=1}^{n} w_{rk} s_r, \sum_{r'=1}^{n} w_{r'l} s_{r'}) \\
&\overset{(b)}{=} \sum_{klqq'rr'}^{n} w_{mk} w_{ml} cum(w_{qi} s_q, w_{q'j} s_{q'}, w_{rk} s_r, w_{r'l} s_{r'}) \\
&\overset{(c)}{=} \sum_{klqq'rr'}^{n} w_{mk} w_{ml} w_{qi} w_{q'j} w_{rk} w_{r'l} cum(s_q, s_{q'}, s_r, s_{r'}) \\
&\overset{(d)}{=} \sum_{klq}^{n} w_{mk} w_{ml} w_{qi} w_{qj} w_{qk} w_{ql} cum(s_q, s_q, s_q, s_q) \\
&\overset{(e)}{=} \sum_{lq}^{n} w_{ml} w_{qi} w_{qj} w_{ql} kurt(s_q) \sum_{k} w_{mk} w_{qk} \\
&\overset{(f)}{=} \sum_{q}^{n} w_{qi} w_{qj} kurt(s_q) \delta_{mq} \sum_{l} w_{ml} w_{ql} \\
&= \sum_{q}^{n} w_{qi} w_{qj} kurt(s_q) \delta_{mq} \delta_{mq} \\
F_{i,j}(w_m w_m^T) &= w_{mi} w_{mj} kurt(s_m)
\end{aligned} \tag{56}$$

On equation 56, we computed the linear transformation of matrix M under the 4th order cumulant $\mathcal{C}_4^Z$. On (a) we applied the linear relation of the ICA model $z = W^T s$, each row of W^T mixed the independent components $z_i = \sum_q w_{qi} s_q$. On (b) we applied the sum property of cumulants. On (c) we applied the scaling property of cumulants. On (b) the independence of random variables s_q, only joint cumulants with the same index $q = q' = r = r'$ are different than zero. On (e) we associated the coefficients with k index and row orthogonality of W lead $\sum_k w_{mk} w_{qk} =$. On (f) we associated the coefficients

with l index which lead $\sum_l w_{mk} w_{qk} = \delta_{mq}$.

The derivation on equation 56 showed that a matrix of the form $M = w_m w_m^T$ is an eigen matrix of the tensor cumulant $\mathcal{C}_4^Z$, and the eigenvalue is the kurtosis of the mth independent component.

The linear transformation $F(M)$ showed that the separating matrix W could be computed by means of the eigen matrices $M = W_M W_m^T$ of the cumulant tensor $\mathcal{C}_4^Z$. Each eigen matrix $w_m w_m^T$ reveal a column of the separating matrix W.

Remark. The solution for the ICA model using tensor eigen decomposition of cumulant was developed parallel by [13]. In that approach they exploited the same eigen vector property inherited by the cumulant tensor.

We established a relation between the linear transformation $F(M)$ and the n-mode product proposed on [31], [15]. We can write the linear transformation on equation 56 as:

$$F(M) = F(w_1 w_1^T) = \lambda M = F \times_3 w_1^T \times_4 w_1^T \tag{57}$$

Note that the cumulant tensor is super symmetric (tensor elements are invariable under index permutation). Exploiting the tensor symmetry left and right eigen matrices M are the same. The linear transformation transposed $F^T(M)$ could be rewrite using mode-n product.

$$F^T(M^T) = F^T(w_1 w_1^T) = \lambda M^T = F \times_1 w_1^T \times_2 w_1^T \tag{58}$$

Equations 57, 58 revealed the eigen vectors of cumulant tensor $\mathcal{C}_4^Z$ are the w_i (columns of unmixing matrix W).

Cumulant tensor $\mathcal{C}_4^Z$ can be expressed by a PARAFAC decomposition [31], [15] according to:

$$F = \sum_{i=1}^{n} \lambda_i w_i \circ w_i \circ w_i \circ w_i = \Lambda \times_1 W \times_2 W \times_3 W \times_4 W \tag{59}$$

On equation 59 we note multiplication of tensor F by W^T on each mode will lead us to a super diagonal tensor Λ. This result represents a cumulant tensor with fourth order joint cumulants equal to zero, i.e. no dependency between variables.

4.4 High order Partial Least Squares

High order Partial Least Squares(HOPLS) [65] is a multilinear regression model. HOPLS predicts a tensor (multiway array) $\underline{Y}$ from a tensor $\underline{X}$ through projecting the data onto the latent space T and performing regression on the corresponding latent variables. HOPLS has two parameters to describe model complexity and fitness; the number of latent variables $t_r \in \mathbb{R}^{I_1 \times 1}$, $r \in \{1, \ldots, R\}$ (also called latent vectors, score vectors, components, scores or data projection $U^T X$ in PCA) and the number of subspace basis vectors L_N inside each factor matrix $P^{(n)} \in \mathbb{R}^{I_n \times L_n}$, $n \in \{1, \ldots, N\}$ (also called direction vectors, loadings, matrix U in PCA).

The low-dimensional latent space is optimized sequentially via a deflation operation, yielding the best joint subspace approximation for both X and Y.

The data is modeled as a sum of R orthogonal Tucker tensors. Compared to standard PLS, HOPLS performs a higher order singular value decomposition on a generalized cross-covariance tensor.

4.4.1 Previous works on HOPLS

The N-way PLS (N-PLS) decomposes the independent X and dependent Y data into rank-one tensors, subject to maximum pairwise covariance of the latent vectors. This promises enhanced stability, resilience to noise, and intuitive interpretation of the results [5], [6]. Due to these desirable properties, N-PLS has found applications in areas ranging from chemometrics [21], [45], [68] to neuroscience [44], [1]. A modification of the N-PLS and the multiway covariates regression was studied in [7], [52], [35], where the weight vectors yielding the latent variables are optimized by the same strategy as in N-PLS, resulting in better fitness to independent data X while maintaining no difference in predictive performance. The tensor decomposition used within N-PLS is canonical decomposition/parallel factor analysis (CANDECOMP/PARAFAC or CP) [20], which makes N-PLS inherit both the advantages and limitations of CP [51]. These limitations are related to poor fitness ability, computational complexity, and slow convergence when handling multivariate dependent data and higher order ($N > 3$) independent data, causing N-PLS to not be guaranteed to outperform standard PLS [23], [31].

4.4.2 Method

HOPLS is a generalized multilinear regression model, called higher order partial least squares (HOPLS), which makes it possible to predict an Mth-order tensor $\underline{Y}$ ($M \geq 3$) (or a particular case of two-way matrix Y) from an Nth-order tensor $\underline{X}$ ($N \geq 3$) by projecting tensor $\underline{X}$ onto a low-dimensional common latent subspace. The latent subspaces are optimized sequentially through simultaneous rank-$(1, L_2, \cdots, L_N)$ approximation of $\underline{X}$ and rank-$(1, K_2, \cdots, K_M)$ approximation of $\underline{Y}$ (or rank-one approximation in particular case of two-way matrix Y). Due to the better fitness ability of the orthogonal Tucker model as compared to CP [31] and the flexibility of the block Tucker model [14], the analysis and simulations show that HOPLS proves to be a promising multilinear subspace regression framework that provides not only an optimal trade off between fitness and model complexity but also enhanced predictive ability in general. HOPLS have a closed-form solution by employing higher order singular value decomposition (HOSVD) [15], which makes the computation more efficient than the classical iterative procedure.

Consider an Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1, \cdots, I_N}$ and an Mth-order tensor $\underline{Y} \in \mathbb{R}^{J_1, \cdots, J_M}$, both tensors have the same size on the first mode $I_1 = J_1$ (same number of samples). The objective of HOPLS is to find an optimal subspace approximation of $\underline{X}$ and $\underline{Y}$, in which latent vectors from $\underline{X}$ and $\underline{Y}$ have maximum pairwise covariance. The problem boils down to find the common latent subspace which can approximate both $\underline{X}$ and $\underline{Y}$ simultaneously.

In order to find the common latent subspace a block-wise orthogonal Tucker decomposition was employed. Tensor $\underline{X}$ was decomposed as a sum of rank-1, $L_2, \cdots, L_N$ Tucker blocks, $\underline{Y}$ was decomposed as a sum of rank-1, $K_2, \cdots, K_M$ Tucker blocks which can be expressed as

$$\begin{aligned} \underline{X} &= \sum_{r=1}^{R} \underline{G}_r \times_1 t_r \times_2 P_r^{(2)} \times_3 \cdots \times_N P_r^{(N)} + \underline{E}_R \\ \underline{Y} &= \sum_{r=1}^{R} \underline{D}_r \times_1 t_r \times_2 Q_r^{(2)} \times_3 \cdots \times_N Q_r^{(N)} + \underline{F}_R \end{aligned} \tag{60}$$

where R is the number of latent vectors, $t_r \in \mathbb{R}^{I_1}$ is the rth latent vector, loading matrices $P_r^{(n)} \in \mathbb{R}^{I_n \times L_n}$, with $n \in \{2, \cdots, N\}$, and $Q_r^{(m)} \in \mathbb{R}^{J_m \times K_m}$, with $m \in \{2, \cdots, M\}$. Core tensors $\underline{G}_r \in \mathbb{R}^{1 \times L_2 \times \cdots \times L_N}$ and $\underline{D}_r \in \mathbb{R}^{1 \times K_2 \times \cdots \times K_N}$.

Decomposition in 60 is not unique due to permutation, rotation and scaling issues [31]. To alleviate this problem, additional constraints should be imposed such that the core tensors $\underline{G}_r$ and $\underline{D}_r$ are all-orthogonal (orthogonal sub tensors), loading matrices are column-wise orthonormal $P_r^{(n)T} P_r^{(n)} = I_{L_n}$ and $Q_r^{(m)T} Q_r^{(m)} = I_{K_n}$, latent vector is of length one. $||t_r||_F = 1$. Thus each summation term in 60 is represented as an orthogonal Tucker model, implying essentially uniqueness as it is subject only to trivial indeterminacies [14].

4.4.3 Optimization criteria and Algorithm

In order to extract latent components we used the sequential method according to [65], which extracts one component at a time. Sequential method compute a latent vector from the multiway array, then deflates the tensor and compute the next one from the residual. The tensor decomposition used for $\underline{X}$ and $\underline{Y}$ was the orthogonal block Tucker model having a common latent component on a specific mode.

The subspace approximation by block Tucker model on 60 aimed to find a set of orthogonal loadings $P_r^{(n)}$, $Q_r^{(m)}$, and latent vectors t_r under certain constraints. Each term can be optimized sequentially with the same criteria based on deflation, the problem simplify to find the first latent vector t and two sequences of loading matrices $P^{(n)}$ and $Q^{(m)}$.

To quantify the quality of the subspace approximation, we used the Frobenius norm of the residuals E and F. Here we present some basic results about norm of core tensor and residuals, necessary to find the latent vectors.

Optimum Core tensor for orthogonal tucker model approximation under Frobenius norm. Given a tensor $\underline{X} \in \mathbb{R}^{I_1, \cdots, I_N}$ and column orthonormal matrices $P^{(n)} \in \mathbb{R}^{I_n \times L_n}$, $n = 2, \cdots, N$, a latent unit norm vector $t \in \mathbb{R}^{I_1}$, $||t||_F = 1$.

$$\underset{\underline{G}}{argmin} \quad ||\underline{X} - \underline{G} \times_1 t \times_2 P^{(2)} \times_3 \cdots \times_N P^{(N)}||_F^2 = \underline{X} \times_1 t^T \times_2 P^{(2)T} \times_3 \cdots \times_N P^{(N)T} \quad (61)$$

Proof is widely used in the literature [31], [15].

Minimization of residual norm is equivalent to maximization of core tensor norm. Given a tensor $\underline{X} \in \mathbb{R}^{I_1, \cdots, I_N}$, the following two constrained optimization problems are equivalent:

$$\begin{aligned} &\underset{\{P^{(n)}, t, \underline{G}\}}{min} ||\underline{X} - \underline{G} \times_1 t \times_2 P^{(2)} \times_3 \cdots \times_N P^{(N)}||_F^2 \\ &s.t. \quad P^{(n)T} P^{(n)} = I_{L_n}, ||t||_F = 1 \end{aligned} \quad (62)$$

$$\begin{aligned} &\underset{\{P^{(n)}, t\}}{max} ||\underline{X} \times_1 t^T \times_2 P^{(2)T} \times_3 \cdots \times_N P^{(N)T}||_F^2 \\ &s.t. \quad P^{(n)T} P^{(n)} = I_{L_n}, ||t||_F = 1 \end{aligned} \quad (63)$$

Proof is available at [31] pp. 477-478

Covariance tensor norm $|| \langle \underline{G}, \underline{D} \rangle_{\{1;1\}} ||_F^2$ is equivalent to the product of each tensor norm $||\underline{G}||_F^2 \cdot ||\underline{D}||_F^2$. Let $\underline{G} \in \mathbb{R}^{1,L_2,\cdots,L_N}$ and $\underline{D} \in \mathbb{R}^{1,K_2,\cdots,K_N}$, then the norm fo the covariance tensor is

$$\begin{aligned} || \langle \underline{G}, \underline{D} \rangle_{\{1;1\}} ||_F^2 &\overset{(a)}{=} ||vec(\underline{G})vec(\underline{D})^T||_F^2 \\ &\overset{(b)}{=} tr(vec(\underline{D})vec(\underline{G})^T vec(\underline{G})vec(\underline{D})^T) \\ &\overset{(c)}{=} ||vec(\underline{G})||_F^2 \cdot ||vec(\underline{D})||_F^2 \end{aligned} \tag{64}$$

Equality (a) by the definition of covariance tensor, (b) by transpose property $||A||_F^2 = tr(A^T A)$. On (c) the cyclic permutation property of trace was applied $tr(ABCD) = tr(BCDA)$.

The goal is to compute a simultaneous block Tucker decomposition for $\underline{X}$ and $\underline{Y}$ such that have minimum residuals $\underline{E}$, $\underline{F}$ respectively under Frobenius norm. By Proposition 2, the minimum residuals are found when the core tensors norms $\underline{G}$ and $\underline{D}$ are maximized. The parameters for the maximization problem are the factor matrices $P^{(n)}$, $Q^{(m)}$ and the common latent vectors t_r.

In order to compute the simultaneous decomposition, [65] proposed to maximize the product of the core tensors norms.

$$\begin{aligned} &max\{||\underline{G}||_F^2 \cdot ||\underline{D}||_F^2\} \\ &s.t. \quad P^{(n)T}P^{(n)} = I_{L_n}, t^T t = 1 \end{aligned} \tag{65}$$

This objective function, according to Proposition 3, is the norm of a mode-n tensor-tensor product or the covariance tensor $\underline{C} = COV_{\{1;1\}}(\underline{G}, \underline{D}) = \langle \underline{G}, \underline{D} \rangle_{\{1;1\}}$. By substitution of the core tensor (equation 61) on the objective function (equation 65), we get

$$\begin{aligned} ||\underline{G}||_F^2 \cdot ||\underline{D}||_F^2 &= || \langle \underline{G}, \underline{D} \rangle_{\{1;1\}} ||_F^2 \\ &= \left|\left| \left\langle [\![\underline{X}; t^T, P^{(2)T}, \cdots, P^{(N)T}]\!], [\![\underline{Y}; t^T, Q^{(2)T}, \cdots, Q^{(M)T}]\!] \right\rangle_{\{1;1\}} \right|\right|_F^2 \\ &= \left|\left| [\![\langle \underline{X}, \underline{Y} \rangle_{\{1;1\}}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!] \right|\right|_F^2 \\ &= \left|\left| [\![\underline{C}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!] \right|\right|_F^2 \end{aligned} \tag{66}$$

Note this form is similar to the optimization problem for two-way PLS in 21, where the cross-covariance matrix $X^T Y$ is replaced by $\langle \underline{X}, \underline{Y} \rangle_{\{1;1\}}$.

Finally the optimization problem is defined as

$$\begin{aligned} &\underset{P^{(n)},Q^{(m)}}{max} \left|\left| [\![\underline{C}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!] \right|\right|_F^2 \\ &s.t. \quad P^{(n)T}P^{(n)} = I_{L_n}, Q^{(m)T}Q^{(m)} = I_{K_n} \end{aligned} \tag{67}$$

where $P^{(n)}$, $n = 2, \cdots, N$ and $Q^{(m)}$, $m = 2, \cdots, M$ are the parameters to optimize. The optimization problem on 67, is equivalent to find the best subspace approximation of $\underline{C}$

$$\underline{C} \approx [\![\underline{G}^{(C)}; P^{(2)}, \cdots, P^{(N)}, Q^{(2)}, \cdots, Q^{(M)}]\!] \tag{68}$$

which can be obtained by rank-$(L_2, \cdots, L_N, K_2, \cdots, K_M)$ HOSVD on tensor $\underline{C}$. According to Proposition 1, the optimization term on 67 is equivalent to the norm of the core $\underline{G}^{(C)}$, in order to maximize it HOSVD should be performed.

Higher order orthogonal iteration (HOOI) algorithm [31], [16], which is known to converge fast, is employed to find parameters $P^{(n)}$ and $Q^{(m)}$ by orthogonal Tucker decomposition of $\underline{C}$.

After loading matrices were found, we can compute the latent vector t. Given a set of loading matrices $\{P^{(n)}\}$, latent vector t should explain variance of $\underline{X}$ as much as possible, that is

$$t = \underset{t}{argmin} ||\underline{X} - [\![\underline{G}; t, P^{(2)}, \cdots, P^{(N)}]\!]||_F^2 \tag{69}$$

which is achieved by choosing t as the first leading singular vector of the unfolding matrix $(\underline{X} \times_2 P^{(2)T} \times_3 \cdots \times_N P^{(N)T})_{(1)}$ as used in the HOOI algorithm [31], [30]. Then, $\underline{G}$ and $\underline{D}$ can be computed using 61.

The procedure should be repeated performing deflation and extracting one latent vector and a set of factor matrices each time, until R components or a residual threshold. HOPLS algorithm is outlined on algorithm 2.

Algoritmo 2: High Order Partial Least Squares

Input : $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\underline{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, $N \geq 3$, $M \geq 3$ and $I_1 = J_1$. Number of latent vectors R and number of loading vectors L_n, K_n for loading matrices $P^{(n)}$ and $Q^{(m)}$.

Output: Loading matrices $\{P_r^{(n)}\}, \{Q_r^{(m)}\}, \{\underline{G}_r\}, \{\underline{D}_r\}, T$, $r = 1, \cdots, R$; $n = 2, \cdots, N$; $m = 2, \cdots, M$

Initialization: $\underline{E}_1 \leftarrow \underline{X}$, $\underline{F}_1 \leftarrow \underline{Y}$

for $r = 1$ *to* R *do* **do**

  if $||\underline{E}_r||_F \geq \epsilon$ *and* $||\underline{F}_r||_F \geq \epsilon$ **then**

    $\underline{C}_r \leftarrow \langle \underline{E}_r, \underline{F}_r \rangle_{\{1,1\}}$

    Orthogonal Tucker Decomposition of $\underline{C}_r$ with rank-$L_2, \times, L_N, K_2, \cdots, K_M$ by HOOI $\underline{C}_r \approx [\![\underline{G}_r^{(C_r)}; P_r^{(2)T}, \cdots, P_r^{(N)T}, Q_r^{(2)T}, \cdots, Q_r^{(M)T}]\!]$

    $t_r \leftarrow$ first leading left singular vector by SVD$\left[\left(\underline{E}_r \times_2 P^{(2)T} \times_3 \cdots \times_N P^{(N)T}\right)_{(1)}\right]$ $\underline{G}_r \leftarrow [\![\underline{E}_r; t_r^T, P_r^{(2)T}, \cdots, P_r^{(N)T}]\!]$

    $\underline{D}_r \leftarrow [\![\underline{F}_r; t_r^T, Q_r^{(2)T}, \cdots, Q_r^{(M)T}]\!]$

    Deflation:

    $\underline{E}_{r+1} \leftarrow \underline{E}_r - [\![\underline{G}_r; t_r, P_r^{(2)T}, \cdots, P_r^{(N)T}]\!]$

    $\underline{F}_{r+1} \leftarrow \underline{F}_r - [\![\underline{D}_r; t_r, Q_r^{(2)T}, \cdots, Q_r^{(M)T}]\!]$

  else

    Break

4.5 HOPLS image classification

Design a image classifier based on features extracted from multilinear structure of input data (image ensemble and labels). The method is based on **tensor decomposition** which enables to **preserve the inter pixel structure on sample images** in contrast to matrix decomposition. The algorithm uses the High Order Partial Least Squares (HOPLS) in order to find a **multilinear subspace approximation of input data.** HOPLS seeks for a representation of **input data** (image samples) which is **maximally correlated** to a linear transformation of **output data** (class labels).

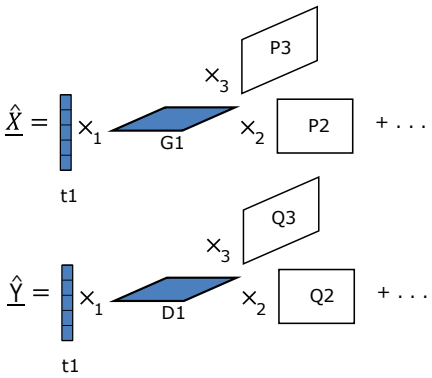
4.5.1 Project Latent Space

$$
\begin{aligned}
X &= TP^T + E = \sum_{r=1}^{R} t_r p_r^T + E \\
Y &= UQ^T + F = \sum_{r=1}^{R} u_r q_r^T + F \\
U &= TD + Z \\
\\
Y &= TDQ^T + (ZQ^T + F) = TDQ^T + F^*
\end{aligned}
$$

4.5.2 High Order Partial Least Squares

$$
\begin{aligned}
\underline{X} &= \sum_{r=1}^{R} \underline{G}_r \times_1 t_r \times_2 P_r^{(2)} \times_3 \cdots \times_N P_r^{(N)} + E \\
\underline{Y} &= \sum_{r=1}^{R} \underline{D}_r \times_1 t_r \times_2 Q_r^{(2)} \times_3 \cdots \times_N Q_r^{(N)} + F
\end{aligned}
\tag{70}
$$

Minimize error approximation



$$
\begin{aligned}
&\underset{\{P^{(n)},t,\underline{G}\}}{min} ||\underline{X} - \underline{G} \times_1 t \times_2 P^{(2)} \times_3 \cdots \times_N P^{(N)}||_F^2 = ||E||_F^2 \\
&s.t. \quad P^{(n)T} P^{(n)} = I_{L_n}, ||t||_F = 1
\end{aligned}
\tag{71}
$$

Equivalent to

$$
\begin{aligned}
&\underset{\{P^{(n)},t\}}{max} ||\underline{X} \times_1 t^T \times_2 P^{(2)T} \times_3 \cdots \times_N P^{(N)T}||_F^2 = ||\underline{G}||_F^2 \\
&s.t. \quad P^{(n)T} P^{(n)} = I_{L_n}, ||t||_F = 1
\end{aligned}
\tag{72}
$$

Minimize both errors $||E||_F^2$ and $||F||_F^2$ is equivalent to maximize the product of each tensor norm.

$$||\underline{G}||_F^2 \cdot ||\underline{D}||_F^2 = ||\left\langle \underline{G}, \underline{D} \right\rangle_{\{1;1\}}||_F^2 = ||\underline{G}^{(\underline{C})}||_F^2 \tag{73}$$

$$||\underline{G}||_F^2 \cdot ||\underline{D}||_F^2 = ||\left\langle \underline{G}, \underline{D} \right\rangle_{\{1;1\}}||_F^2 \tag{74}$$

$$\begin{aligned} &\left|\left|[\![\langle \underline{X}, \underline{Y} \rangle_{\{1;1\}}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!]\right|\right|_F^2 \\ &= \left|\left|[\![\underline{C}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!]\right|\right|_F^2 \end{aligned} \tag{75}$$

$$\begin{aligned} &\underset{P^{(n)}, Q^{(m)}}{max} \left|\left|[\![\underline{C}; P^{(2)T}, \cdots, P^{(N)T}, Q^{(2)T}, \cdots, Q^{(M)T}]\!]\right|\right|_F^2 \\ &s.t. \quad P^{(n)T} P^{(n)} = I_{L_n}, Q^{(m)T} Q^{(m)} = I_{K_n} \end{aligned} \tag{76}$$

Under the orthogonal factor matrix constraint, the former maximization seeks to find the best subspace approximation of C of rank $(L_2, \cdots, L_n, K_2, \cdots, L_m)$

4.5.3 Results

The proposed methodology was applied to classify images on two datasets. The Labeled Faces in the Wild (LFW) and the Weizmann Face database, following the experimental guidelines suggested on [23]. The experiment aimed to classify faces samples of three people. The LFW data set consisted of 705 images with background for the three persons. Cross validation partitions was performed to test the performance. The training set was arranged as a third order tensor X, each sample image on rows. Its dimensions were samples $\times$ pixels width $\times$ pixels height.A matrix Y of size samples $\times$ classes, was build by assigning a 1 on the element corresponding to each class. The training set was used to compute model parameters $\{P_r^{(n)}\}$,$\{Q_r^{(m)}\}$,$\{\underline{G}_r\}$, $\{\underline{D}_r\}, T$, $r = 1, \cdots, R$; $n = 2, \cdots, N$; $m = 2, \cdots, M$. Predictions Y of new samples were used to perform image classification.

A grid search map was build to find the best number of parameters. On figure 13 the inverse of error prediction was presented. This figure showed the best parameters for the decomposition are 8 loadings vectors on each mode matrix and 40 tucker blocks.

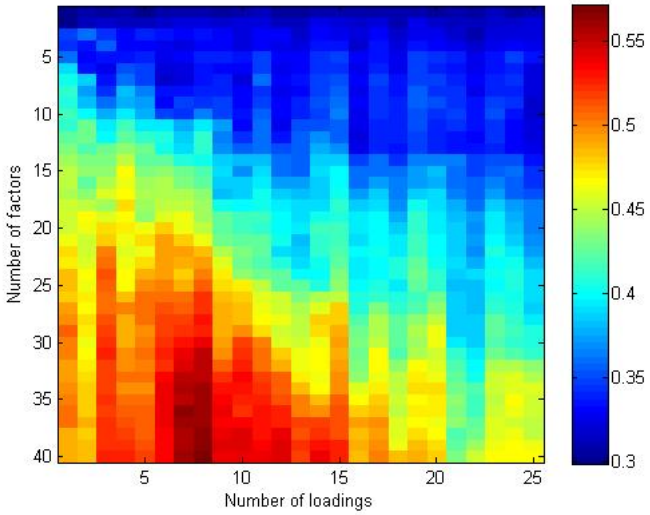


Figure 13: Parameters grid search.

On figure 14, was depicted the scatter plot for the predictions of images on the three classes represented by red, green and blue dots. It can be noticed that points representing images were overlapping, this reflected a poor separability performance. On figure 15, was shown the predictions of images labels using the proposed algorithm. The scatter plot pictures dots representing images forming clusters by color, representing a better separability performance.

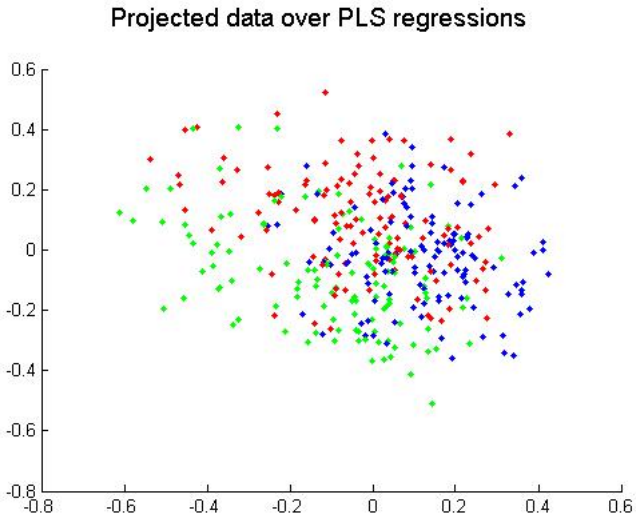


Figure 14: Prediction of labels using PLS method.

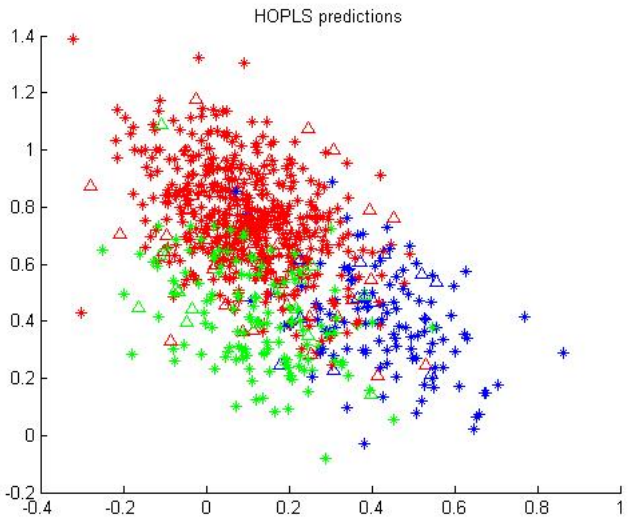


Figure 15: Prediction of labels using the proposed method.

The results for the proposed method on a 10-fold cross validation was depicted on table 1.

Table 1: LFW, HOPLS, factor 40, loadings 8, 10.6 seg

Accuracy			
Training	Test		
max	max	kNN	Random Forest
0.85175	0.7875	0.8125	0.8375
0.84895	0.85	0.825	0.8375
0.84218	0.78481	0.81013	0.78481
0.8352	0.77215	0.79747	0.78481
0.86034	0.74684	0.77215	0.75949
0.84358	0.74684	0.75949	0.74684
0.8352	0.8481	0.83544	0.82278
0.84476	0.7	0.7625	0.7625
0.81958	0.7625	0.8	0.7875
0.84755	0.7625	0.725	0.7875

It can be noted that the extracted subspace has defined patterns able describe a face and shoulders of a person. Additionally its seen that spatial relations are reflected on the extracted subspace. This can be noted on the comparison between subspaces extracted by matrix and tensorial method on figure 16.

Figure 16: Features learned by the proposed method.

The results on a benchmark comparison with similar linear subspace approximation method was depicted on table 2. Input: Image ensemble. Output: Vector class labels (One hot).

Table 3: Additional HOPLS classification results

Dataset	Precision
LFW	84.81%
Tensor Faces	98.33%
BAD	60.03%
Sports Action	38.61%
Sports Action + Gabor Filter	47.22%
Silhouette	54.72%

Table 2: Accuracy by method for LFW

Method	Accuracy
HOPLS	84.81
PCA	34.62
PCA class	31.68
PLS	54.76
DLDA (Direct LDA) [Yu and Yang, 2001]	
PILDA (Pseudoinverse LDA) [Tian et al. 1986]	82.50
FPILDA (Fast PILDA) [Liu et al., 2007]	82.50
PCAplusLDA	75.95
NLDA (Null LDA) [Chen et al., 2000]	86.25
OLDA (Orthogonal LDA) [Ye 2005]	83.75
ULDA (Uncorrelated LDA) [Ye et al., 2004]	82.50
QRNLDA (QR based NLDA) [Chu and Thye, 2010]	83.75
FNLDA (Fast NLDA) [Sharma and Paliwal, 2012]	83.75
CLDA (Discriminant common vector LDA) [Cevikalp et al., 2005]	83.75
IPILDA (Improved PILDA) [Paliwal and Sharma, 2012]	83.75
ALDA (Approximate LDA) [Paliwal and Sharma, 2011]	66.25
EFR (Eigenfeature Regularization) [Jiang et al., 2008]	88.75
ELDA (Extrapolation LDA) [Sharma and Paliwal, 2010]	85.00
MLDA (Maximum Uncertainty LDA) [Li et al., 2003]	83.75
IDLDA (Improved DLDA) [Paliwal and Sharma, 2010]	66.25
TSLDA (Two-stage LDA) [Sharma and Paliwal, 2012a]	
Range_Sw (Range space of within-class scatter matrix)	61.25
Null_Sw (Null space of within-class scatter matrix)	83.75
Range_Sb (Range space of between-class scatter matrix)	58.75
Null_Sb (Null space of between-class scatter matrix)	58.75

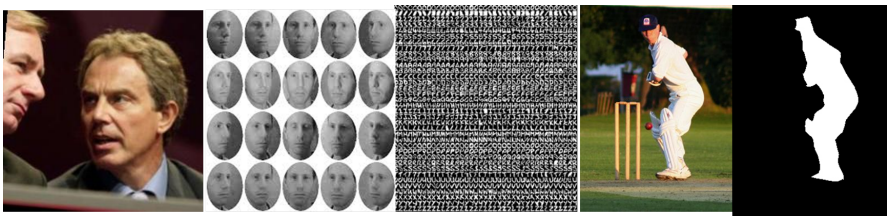


Figure 17: Test Datasets

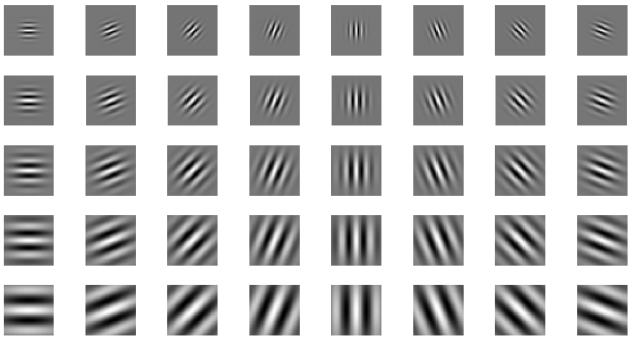


Figure 18: Gabor Filters real components

5 Solutions and approaches made by other authors

On computer vision research area, several approaches exist to perform feature extraction and classification. This proposal focuses on multilinear algebra approaches to accomplish such tasks, specifically on subspace approximation. The most relevant approaches include: linear and multilinear subspace approximation for feature generation (SVD [46], Tucker decomposition [56]), joint subspace approximation (PLS [56], HOPLS [65]), image classification (eigenfaces [28]), non-linear relationship extraction (alpha-beta divergence [42]), and t-SVD tensor completion for image inpainting [53] [64].

In this context, Kembhavi et al. [27] attempted to identify vehicles on aerial images using three types of features, such as Color Probability Maps, Histograms of Oriented Gradients, and Pairs of Pixels. They applied dimensional reduction by Partial Least Squares (`PLS`) and classification by a linear classifier. The concatenation of the three features leads to a very high-dimensional feature vector (approximately 70,000 elements). In this way, they used PLS analysis to extract a low-dimensional feature subspace, in contrast, with our proposal based on MSL to find features between samples.

Consequently, Zhao et al. [65] introduced a new regression model, termed the Higher-Order Partial Least Squares (`HOPLS`). It aimed to predict a tensor (multiway array) Y from a tensor X through data projection onto a latent space and performing regression on the corresponding latent variables. HOPLS explains the data by a sum of Tucker tensors, which differentiates it substantially from other regression models. In order to decompose X and Y simultaneously, a higher-order singular value decomposition (`HOSVD`) [56], [31] was also applied on a cross-covariance tensor to optimize the orthogonal loadings. From this work, we take the concept of covariance tensor, and we built it from samples. This tensor preserves the spatial structure. We also assume linear dependency between dimensions, width, height, channels. This constraint was established using the Tucker model.

Vasilescu et al. [58] introduced a multilinear modeling technique to extract factors called Tensor Faces from an ensemble of facial images. They hypothesized that natural images were the composite of multiple factors related to scene structure, illumination, and imaging. The combination of factors was assumed to be linear. The assembled tensor dataset was built by facial images on various modes, including different facial geometries (people), expressions, head poses, and lighting conditions. N-mode SVD was applied to the image ensemble, and the Tensor Faces were extracted. They generated tensor data by arranging images as vectors, losing spatial structure. The tensor spaces were extracted from a core tensor product by a single factor matrix, which yields low-rank images. Additionally, they analyze images as a whole in contrast to our method, where we understand features from patches to obtain more specialized features.

In the Jian Yang et al. work [26], they decompose an image $\mathbf{A}$ in its mode 1, i.e. $\mathbf{A} = \sum_i^M A_i^T \times A_i$, and factors are the eigenvectors, similar to decompose the unfolding on mode 2. They projected all the samples with the factors and built a feature matrix. To classify the test sample, they projected the sample with the factors and applied the nearest neighbor classifier using the distances between the feature matrices on training and test.

For image classification, Fu et al. [18] built a discriminant subspace learning algorithm, called Correlation Tensor Analysis (`CTA`), which incorporate graph-embedded correlational mapping, discriminant analysis, and tensor feature extraction. For the last one, they decompose a tensor built as an array of images using the Tucker model and use the core tensor as features.

On Multilinear Principal Component Analysis (`MPCA`) of Tensor Objects for Recogni-

tion [39], they decompose a tensor built as an array of images using the tucker model to find factor matrices. Then, they project new samples using the factor matrices to find a core tensor Y. In order to perform classification, they computed the distance between training projections and test projection.

On Multilinear Principal Component Analysis Network (`MPCANet`) for Tensor Object Classification [63], from a third order training data tensor, they extracted $I_1 \times I_2 \times I_3$ overlaping tensor patches. Arrange them to build a fourth order tensor $\mathbf{A}$. The resulting tensor $\mathbf{A}$ is decomposed with a Tucker model to find factor matrices $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$. Each tensor patch is projected with the factor matrices resulting in $I_1 \times I_2 \times I_3$ tensors $S^1_{m,q}$. The projected tensors $\underline{S}$ are vectorized and truncated building a matrix $\mathbf{Z} \in R^{I_1 I_2 I_3 \times L1}$. The columns of $\mathbf{Z}$ are arranged as tensor to build $F^l, l = 1, ..., L1 \in R^{I_1 \times I_2 \times I_3}$, third-order tensors. The L_1 tensors F^l are the tensor feature output of $\mathbf{X}_m$ for each stage. This construction loses the spatial information, which differs from our implementation because we retain spatial information through the network.

On the other hand, the image classification problem introduces a challenging task due to the large data's intrinsic intra-class variability. This variability arises from differences in lightning, misalignment, non-rigid deformations, occlusion, and corruptions [10]. A relevant effort to counter this issue is the Wavelet Scattering Network (`ScatNet`), which prefixes the convolutional filters [8], [49]. They are simply wavelet operators; hence no learning is needed at all [10]. Inspired by these results, Tsung-Han Chan et al. [10] proposes the PCA Network (`PCANet`). This network uses PCA to generate data-adapting convolutional filter banks, which are prefixed in each convolutional layer. The process to generate filters is patch extraction, vectorization, covariance matrix, eigenvectors calculation, normalization, and conversion to two-dimensional kernels. This architecture adds non-linear layers as binary quantization (hasting) and the feature pooling layer as block-wise histograms. Finally, the author suggests that this method could be applied in cascade style, using many layers in a deeper structure.

In contrast, our method preserves the spatial structure on patches by computing the covariance tensor and Tucker's decomposition. Another difference is that our kernels (filters) are multichannel tensors (third-order tensors), and our experiments suggest our proposed architecture handles multilayer deeper network configurations. Another contrast is that we require a small data subset to cover the spectrum of all the classes coming from the global dataset. This approach compensates for the intra-class variability. Our proposal also uses the well-known ConvNet VGG architecture with Leaky ReLU activation and max pooling, although the patch extraction, prefixed kernels, and normalization steps remain.

Another computer vision task solved by tensor decomposition is image in-painting. Song et al. [53] aimed to recover missing values in images. In their work, their input was a degraded image with random missing pixels. First, they estimate the image using a triangulation-based linear interpolation. Then, they find the most similar candidate patches for each center missing pixel patch according to their Euclidean distance and group them as a tensor. Finally, they painted patches using t-SVD decomposition. Another recent work [64] relies on the painting by a low-rank regularization which minimizes the nuclear tensor norm (TTN) of the patch tensor to alleviate the Patch Mismatch problem. This approach reveals the advantages of representing images as tensors. Such as preserve spatial structure, allow multichannel representation, establish linear dependencies between dimensions and establish representations.

On Hybrid Networks (`HybridNet`) [61], they propose a tensor-factorization (`TFNet`) to alleviate the image vectorization present in PCANet, which destroys spatial image structure. They propose a HybridNet, which is based on both tensor and matrix decom-

positions to obtain feature maps. Like PCANet, they extract image patches and build a third-order tensor, but now they decompose it using Tucker. Then, they compute the filters as the outer product of factor matrices U_1, U_2 columns. They repeat the procedure by inputting the convolved images and using filters from the previous layer for later layers. In this model, they take into account filters extracted using PCA and Tucker decompositions. A drawback is their filters are rank one image and are not able to represent image complexity correctly. In contrast, our proposal computes tensor spaces using a covariance tensor and its factorization, which allows us to generate higher rank filters.

Kossaifi et al. [32] attack the problem generated by flattening at fully-connected layers, which discards multilinear structure in the activations and requires many parameters. They introduce Tensor Contraction Layers (`TCLs`), reducing input layer dimensionality while preserving their multilinear structure. Additionally, Tensor Regression Layers (`TRLs`) transform the regression weights through the factors of a low-rank tensor decomposition. The work presents an end-to-end trainable architecture that retains the multi-modal tensor structure throughout the network.

About tensor operations libraries, [33], [29], [3] proposed software frameworks for tensor computation. Also, they survived the available software for tensor operations. Frameworks include tensor manipulation, products, decompositions, and tensor regression learning [19]. Our work's code was based on the one proposed at [3] wrote in C++.

6 Proposal

6.1 Work Description and Hypothesis

This work describes a new method to extract image features using tensor decomposition to model data. Given a set of sample images, we extract patches from images, compute the covariance tensor for all patches, decompose with the Tucker model, and obtain the most critical features from a tensor core. To extract features, we factorize the covariance tensor (CovTen) into its core and propose a new interpretation of the resultant tensor structure, which holds relevant features in a block-wise arrangement (also named filters, weights, or kernels). This tensorial representation allows preserving the spatial structure, learning multichannel filters, and establishing linear dependence between dimensions, reducing the dimensional complexity (the curse of dimensionality). Thus, the proposed method generates filters by a single feed-forward step using a few samples per class as low as 1. Besides, in kernel generation, labels are not needed. The obtained features were extensively tested using a convolutional neural network for classification. All tests were conducted under the VGG architecture conventions. The experiments helped us identify the proposed method's advantages versus traditional convolutional neural networks in inference capacity and kernels initialization. We also performed experiments to select hyperparameters (nonLinearity, max pooling, samples, filter size) according to their performance. The inference capacity results showed an increased classification accuracy around 67% with CIFAR 10, 64% with CIFAR 100, and 98% with MNIST, using 10,100,1000 samples with a single feed-forward training. On the other hand, the initialization experiments showed the feature extraction capability versus available initializers (He random, He uniform, Glorot, random), confirming linear tensor constraints' usefulness to generate features. Using the method as kernel initializer returns comparable findings with state of the art around 91% with CIFAR 10, 72% with CIFAR 100, and 99% with MNIST.

The backbone of this research relies on well known and widely developed hypothesis about image representation as linear combination of basis vectors [28], [57], [58], [59], [60]. This hypothesis considers there are families of patterns for which it is possible to obtain a useful systematic characterization. It follows that certain family could be low dimensional i.e. any given member might be represented by a small number of parameters [28]. An example of such a family is the human face.

By its nature, data captured by multi sensory technology has a multiway structure, e.g. the dimensions for an RGB image data set are: spatial $\times$ spatial $\times$ color channel $\times$ samples. Such data sets could be naturally represented by multiway arrays (tensors) [11].

The hypothesis is that multilinear algebra and its methods, such as tensor decompositions, capture multiple interactions and couplings on data. Such decompositions are able to discover hidden structure which can not be obtained by traditional matrix methods. This hypothesis is supported by promising results obtained on several works on different areas which include: audio, image and video processing, machine learning, and biomedical applications [11]. Tensor decompositions has well defined applications and benefits, for example: Tucker decomposition [56], [15], [31] minimize the Frobenius norm of the error when a multilinear subspace approximation is performed. On the other side, Canonical decomposition (CANDECOMP) [20], [9] [13] applied over a High Order Statistics tensor, is able to recover independent sources similar to the Independent Component Analysis [24], [25].

Unlike matrices, tensors are multiway arrays of data samples whose representations

are typically overdetermined, therefore fewer parameters in the decomposition than the number of data entries. This gives us an enormous flexibility in finding hidden components in data and the ability to enhance robustness to noise [11].

6.1.1 Hypothesis

- Tensor representation allows to capture hidden structure on data.
- Tensor decomposition generates relevant features for image classification.

6.2 Objectives

- Develop a multilinear subspace approximation method for tensor input data.
  - Develop an algorithm to extract features exploiting inter pixel spatial structure.
  - Develop an algorithm to extract features exploiting class labels.
- Develop an algorithm to classify images using extracted features.
- Develop an algorithm to use generated features under a CNN architecture to perform classification.
- Perform extensive simulations to justify the approach.

6.3 Approach

6.3.1 Covariance Tensor Method

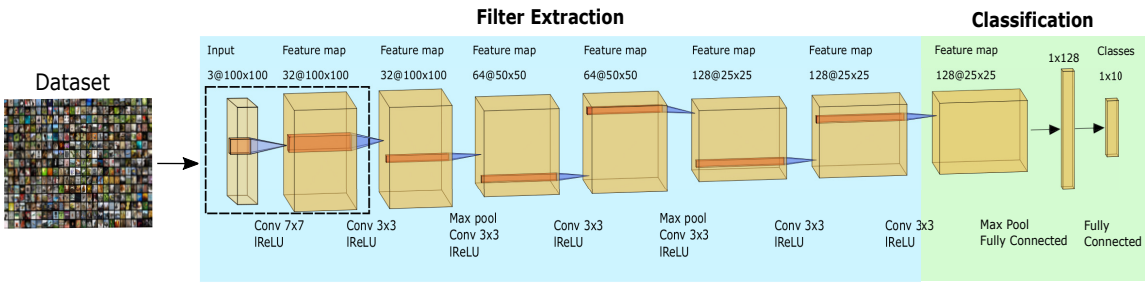


Figure 19: Proposed architecture. For each convolutional layer kernels were generated. Kernel generation was highlighted by dashed lines Figure 27.

The backbone of this research relies on well known and widely developed hypothesis about image representation as linear combination of basis vectors [28], [57], [58], [59], [60]. This hypothesis considers families of patterns for which it is possible to obtain a useful systematic characterization. It follows that certain families could be low dimensional, i.e., any given member might be represented by a small number of parameters [28].

In this regard, our work presents an algorithm to compute kernels from an unlabeled image dataset. We propose to build a covariance tensor and decompose it using the tucker model to find factor matrices. Then, we factorize the covariance tensor and finally extract kernels from the resulting tensor space.

6.3.2 Tensor operations

Starting from a matrix $X \in \mathbb{R}^{I_1 \times I_2}$, which is a two order or two index array, and an Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a higher-dimensional array. The dimensions for an RGB image dataset could be: width × height × color channels × samples. In this regard, such datasets could be naturally represented by multiway arrays (tensors) [11].

The **n-mode unfolding** of a tensor is a matrix denoted by $X^{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$. An unfolding is built by rearranging n-mode fibers as columns of the unfolding matrix $X^{(n)}$.

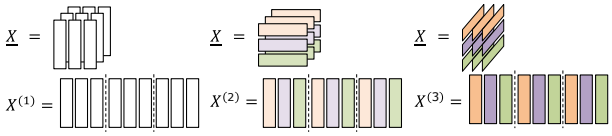


Figure 20: Unfolding of an order three tensor.

The **n-mode product** between a tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and a matrix $U \in \mathbb{R}^{J_n \times I_n}$ is denoted by $\underline{Y} = \underline{X} \times_n U \in \mathbb{R}^{I_1 \times \cdots \times J_n \times \cdots \times I_N}$. The product on mode n requires the number of columns on matrix U be equal to the number of elements on the n-mode fiber from $\underline{X}$.

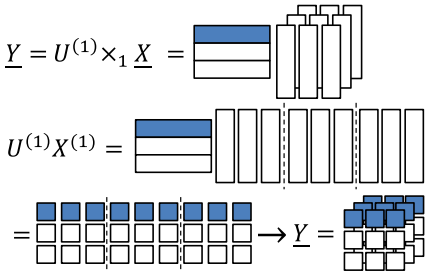


Figure 21: Mode 1 product between a matrix and a third order tensor.

The **n-mode cross covariance** between an Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and an Mth-order tensor $\underline{Y} \in \mathbb{R}^{J_1 \times \cdots \times I_n \times \cdots \times J_N}$ with the same size I_n on the nth mode is defined as Equation 77. For further explanations, we called it covariance tensor.

$$\begin{aligned} &\underline{C} = COV_{\{n;n\}}(\underline{X}, \underline{Y}) = \langle \underline{X}, \underline{Y} \rangle_{\{n;n\}} \\ &COV_{\{n;n\}}(\underline{X}, \underline{Y}) \in \mathbb{R}^{I_1 \cdots I_{n-1}, I_{n+1} \cdots I_N, J_1 \cdots J_{n-1}, J_{n+1} \cdots J_M} \end{aligned} \tag{77}$$

The symbol $\langle \bullet, \bullet \rangle$ represents an **n-mode multiplication** between two tensors. Its element-wise computation was defined on Equation 78.

$$\begin{aligned} &c_{i_1, \cdots, i_{n-1}, i_{n+1}, \cdots, i_N, j_1, \cdots, , j_{n-1}, j_{n+1}, \cdots, j_M} \\ &= \sum_{i_n=1}^{I_n} x_{i_1, \cdots, i_n, \cdots, i_N} y_{j_1, \cdots, i_n, \cdots, j_M} \end{aligned} \tag{78}$$

The computation of the covariance tensor is similar to the computation of sample covariance matrix $\Sigma = XX^T$, where columns $x^{(i)}$ of the matrix X are samples. In that case, the covariance matrix is computed as the sum of outer product $x^{(i)} \circ x^{(i)T}$ for each sample vector, as shown in Figure 22. On the other hand, for the covariance tensor, samples $X^{(i)}$ are multilinear arrays with order $N \geq 2$. In that case, the covariance tensor is computed

as the sum of outer product $X^{(i)} \circ X^{(i)T}$ for each sample, as depicted in Figure 22.

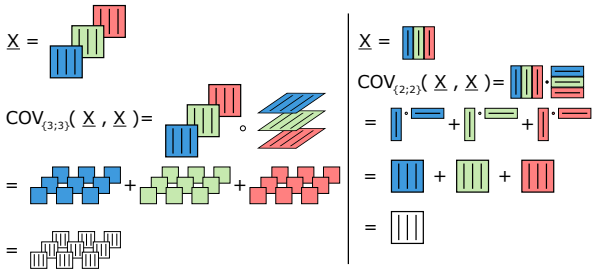


Figure 22: Tensor covariance for $\underline{X} \in \mathbb{R}^{3\times3\times3}$ (left) and $\underline{X} \in \mathbb{R}^{3\times3}$ (right). Samples are frontal frames (mode-3 frame) $X^{(i)} \in \mathbb{R}^{3\times3}$. Resulting covariance tensor has order four and two order respectively.

6.3.3 Covariance Tensor decomposition

Consider a real N-th order tensor $\underline{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$. Tucker decomposition [56] factorizes an input tensor $\underline{X}$ into a core tensor $\underline{G}$ and a set of factor matrices $U^{(n)}$.

$$\begin{aligned} \underline{X} &= \underline{G} \times_1 U^{(1)} \times_2 \cdots \times_N U^{(N)} \\ &= [\![G, U^{(1)}, \cdots, U^{(N)}]\!] \end{aligned} \tag{79}$$

The n-th factor matrix $U^{(n)}$ was build by the left singular vectors of the tensor unfolded on mode n, $X^{(n)} \in \mathbb{R}^{I_n\times I_1\ldots I_{n-1}I_{n+1}\ldots I_N}$.

$$X^{(n)} = U^{(n)}\Sigma^{(n)}V^{(n)T} \tag{80}$$

The factor matrices $U^{(n)}$ are orthogonal, and their transposes are used to compute the core tensor $\underline{G}$.

$$\underline{X} \times_1 U^{(1)T} \times_2 \cdots \times_N U^{(N)T} = \underline{G} \tag{81}$$

Tucker decomposition [56], [15], [31] seeks to minimize the error, using Frobenius norm, between the original tensor and Tucker approximation. An example of covariance tensor from a third-order tensor and its decomposition under the Tucker model is depicted in Figure 23

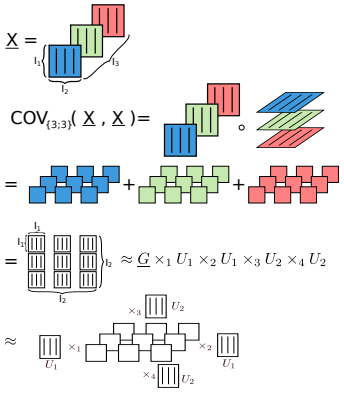


Figure 23: Covariance tensor decomposition for $\underline{X} \in \mathbb{R}^{3\times3\times3}$. Samples are frontal frames (mode-3 frame) $X^{(i)} \in \mathbb{R}^{3\times3}$. Resulting covariance tensor has order four.

6.4 CovTen and CNN

Convolutional Neural Networks are composed of convolutional filters which perform feature extraction followed by fully connected filters in charge to infer the image class. Complete system diagram of a ConvNet was depicted on Figures 19, 24. Our method generates convolutional filters for feature extraction, and the fully connected filters were generated by backpropagation.

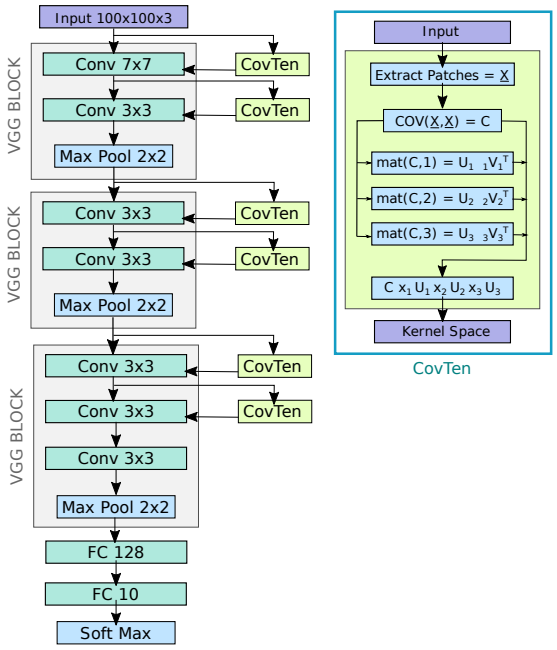


Figure 24: Basic workflow of Covariance Tensor (CovTen) method implemented in a VGG 7 architecture.

As a classification problem, we start from a dataset composed of images of different classes. The design of our method allows us to generate filters without using labels. The goal is to find kernels set $\underline{Y}$ to initialize convolutional layers. Dimensions for a dataset composed by multichannel images are width, height, channels, samples, represented by $\underline{T} \in \mathbb{R}^{w,h,c,s}$. For each sample, we extracted tensor patches using a given size and stride. Patches dimensions are patch-height, patch-width, channels. We subtracted the mean from each one. Tensor $\underline{X}$ was composed using all patches, the first and second dimensions (I_1, I_2) were patch-height and patch-width depicted as rectangles. The third dimension (I_3) represented channels arranged as depth, the fourth dimension (I_4) arranged as rows represented different multichannel patches. We computed the outer product $COV(X, X)$ along the fourth dimension in order to build a sixth-order covariance tensor, following Equation 82. We repeated the whole process for every image, accumulated the covariance tensor, and normalized it by the number of samples.

$$\begin{aligned} &\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4} \\ &\underline{C} = COV(\underline{X}, \underline{X}) \in \mathbb{R}^{I_1 \times I_1 \times I_2 \times I_2 \times I_3 \times I_3} \end{aligned} \tag{82}$$

The resulting covariance tensor is supersymmetric, so we find the odd unfoldings, i.e., modes 1, 3, and 5. For each unfolding, we compute its Singular Value Decomposition (SVD) to find factor matrices U_1, U_3, U_5 following the Tucker model. Finally, we factorize the covariance tensor using the factor matrices yielding tensor $\underline{G}$. We propose to extract covariance filters from the result.

$$
\begin{aligned}
\underline{C}^{(1)} &= U^{(1)}\Sigma^{(1)}V^{(1)T} \\
\underline{C}^{(3)} &= U^{(3)}\Sigma^{(3)}V^{(3)T} \\
\underline{C}^{(5)} &= U^{(5)}\Sigma^{(5)}V^{(5)T} \\
U_1^T &\in R^{J_1 \times I_1} \\
U_3^T &\in R^{J_3 \times I_3} \\
U_5^T &\in R^{J_5 \times I_5}
\end{aligned}
\tag{83}
$$

$$
\underline{C} \times_1 U_1^T \times_3 U_3^T \times_5 U_5^T = \underline{G} \tag{84}
$$

Algoritmo 3: CovTensor to Generate Kernels Space

FuncName: `CovTensorMethod`
Inputs : `Multichannel tensor` $\underline{T}_{w,h,c,s}$, `kernel_size` ks, `stride` st
Output : `Kernel tensor spaces` $\underline{Y}_{k,k,c,n}$
$\underline{C}^{cum}$ = `zero_tensor`
for (`image` I `in` $\underline{T}$) **do**
  $\underline{X}$ = `GetPatches(`I`,`ks`,` st`)`
  $\underline{C}^{cum} = \underline{C}^{cum} + COV(\underline{X}, \underline{X}^T)$
end
$\underline{C} = \frac{\underline{C}^{cum}}{\texttt{NumImg}(\underline{T})}$
$U_1\Sigma_1 V_1^T$ = `SVD(mat(`$\underline{C}, 1$`))`
$U_3\Sigma_3 V_3^T$ = `SVD(mat(`$\underline{C}, 3$`))`
$U_5\Sigma_5 V_5^T$ = `SVD(mat(`$\underline{C}, 5$`))`
$\underline{Y} = \underline{C} \times_1 U_1^T \times_3 U_3^T \times_5 U_5^T$

The factorized tensor is $\underline{G} \in \mathbb{R}^{J_1,I_1,J_2,I_2,J_3,I_3}$. Factor matrices U_1, U_3, U_5 has J_1, J_3, J_5 columns respectively. Factor matrices used for covariance tensor factorization, allow us to choose the amount of filters to generate by selecting the amount of columns J_1, J_2, J_3 from matrices U_1, U_2, U_3. Analyzing the dimensions of the resulting tensor $\underline{G}$, we proposed to rearrange $J_1, I_1, J_2, I_2, J_3, I_3$ as $I_1 \times I_2 \times I_3 \times J_1 J_2 J_3$. Resulting into a fourth order tensor $\underline{Y} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times J_1 J_2 J_3}$, where each multichannel array $I_1 \times I_2 \times I_3$ is an image space and there are $J_1 J_2 J_3$ spaces. Experiments showed us the spatial structure was retained by this reconstruction. Figures 25, 26 reveal a graphical process from patch extraction to kernel generation. Figure 25 depicts the process when the inputs are single channel images. Figure 26 shows the process when the inputs are multiple channel images. In order to generate first layer filters we followed the procedure described in Algorithm 3. In order to generate a convolutional neural network architecture, we have to generate filters for each convolutional layer, see Figure 27. The filter generation procedure Algorithm 3 have to be sequentially repeated for each layer as described on Algorithm 4.

Algoritmo 4: CovTensor in a Convolutional Network

Inputs : `ConvNet architecture, Dataset tensor` $\underline{T}_{w,h,c,s}$, `kernel_size` ks, `stride` st
Output : `ConvNet with initialized kernels`
$\underline{T}^{(input)} = \underline{T}$
for (`layer in ConvNet`) **do**
  $\underline{Y}$ = `CovTensorMethod(`$\underline{T}^{(input)}$`,`ks`,`st`)`
  `InsertKernelsToLayer(layer,`$\underline{Y}$`)`
  $\underline{T}^{(input)}$ = `convolution(`$layer, \underline{T}^{(input)}$`)`
end

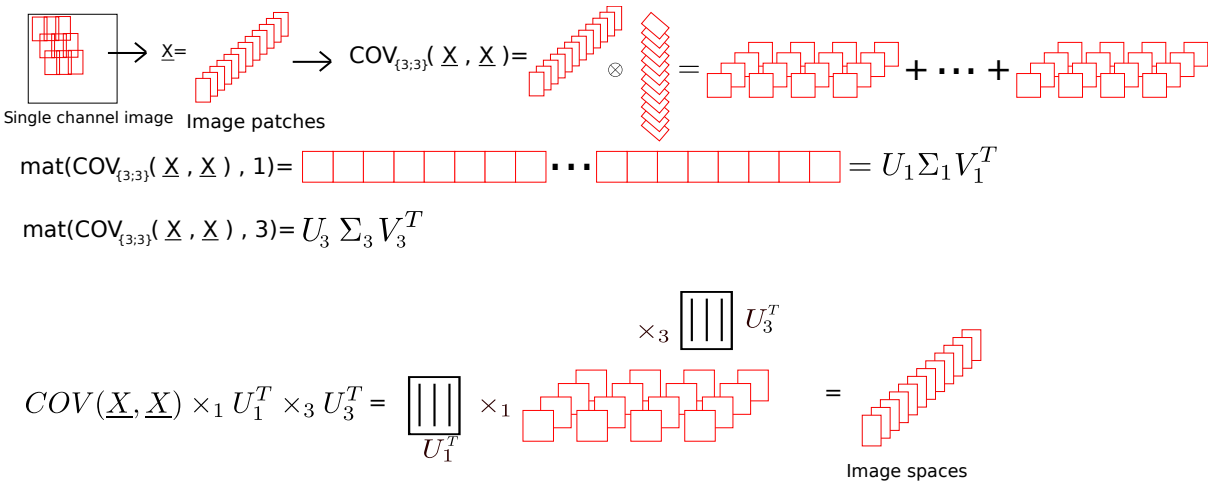


Figure 25: Training stage for a single channel image. The function mat is the unfolding in mode 1 and 3 respectively.

As depicted in Figure 27, the first layer's input is a tensor composed of images (unlabeled data set). We sequentially extract patches, build a patch tensor, generate filters, convolve filters with layer input, generate filter maps and build the following layer patch tensor. Finally, using the feature map from the last layer, fully connected layers were trained to perform classification, as depicted in Figure 19. During inference, we feed-forward all the convolutional layers using a VGG architecture and compute image labels.

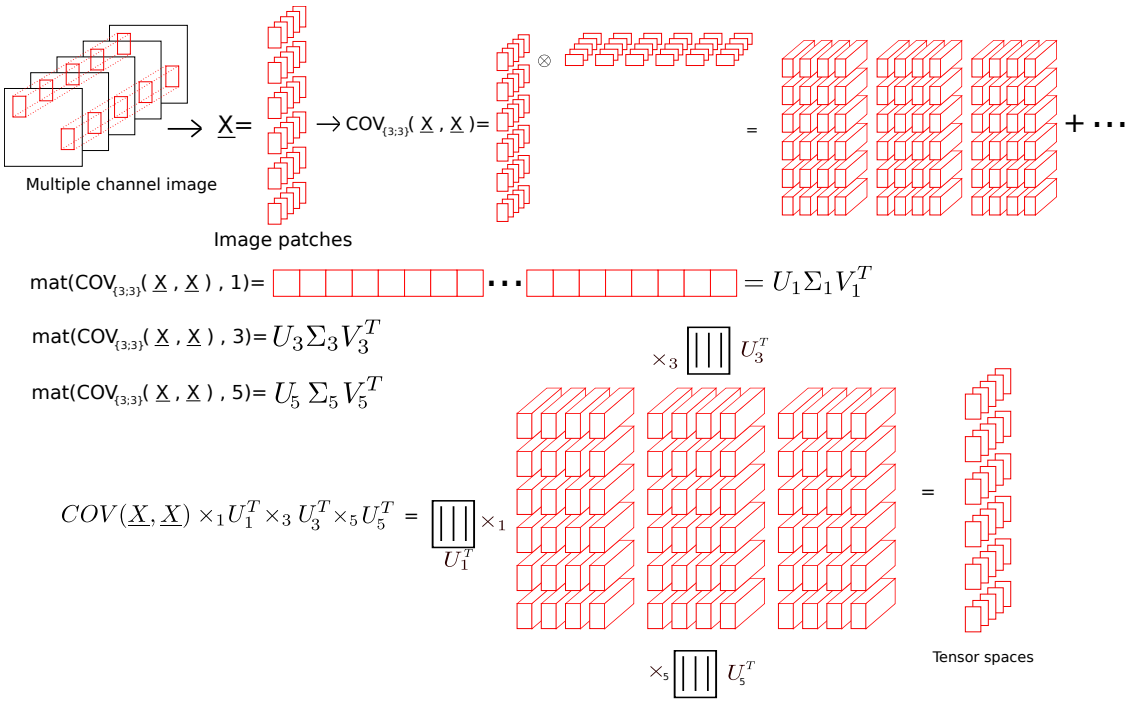


Figure 26: Training tensor stage for a multichannel image. The function mat is the unfolding mode 1, 3 and 5 respectively.

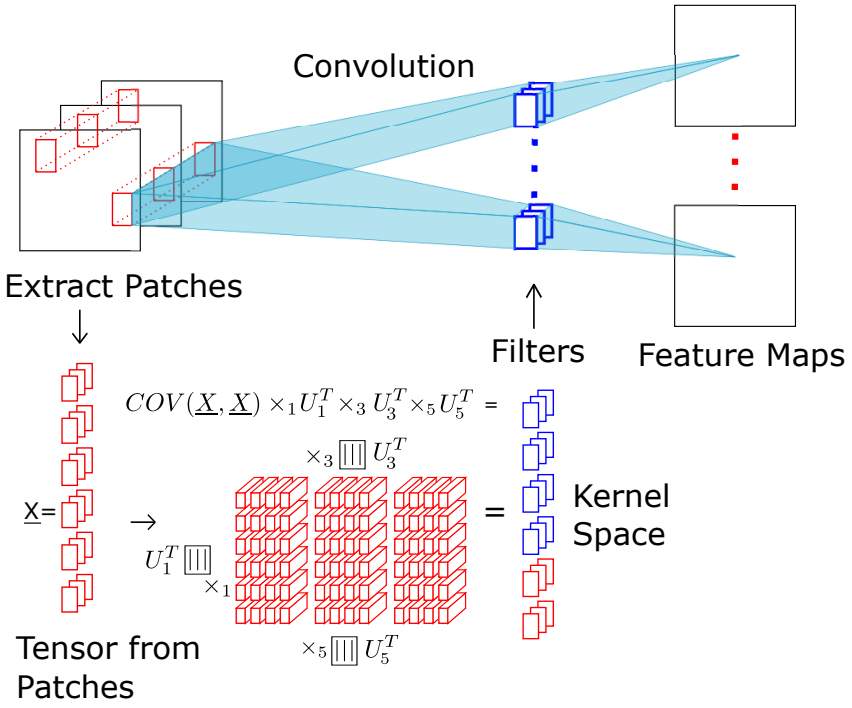


Figure 27: Convolution and kernel space generation.

7 Experimental Results

In this section, we describe the configuration of the experiments and discuss these choices.

Table 4: Network architecture outlines for **CIFAR 10** are used for the configuration and classification experiments (top row), where each network's name matches its respective sections. Their elements are in **bold** when they indicate a frozen layer. Also, it is detailed when a layer use a **Leaky ReLU (+)** or **ReLU (*)** activation, and when an architecture use or no a maxpooling layer.

Hyperparameters Selection							**Classification Experiments**	
Kernel Size		Training Samples Number	Activation and Max Pooling			Kernel and Feature Maps	Inference Capacity	Kernel Initializer
A.1	A.2	B	C.1	C.2	C.3	D	A	B
7 weight layers	7 weight layers	9weight layers	9 weight layers	9 weight layers	9 weight layers	9 weight layers	9 weight layers	9 weight layers
input (32 x 32 RGB image)								
conv3-32 **conv3-32**	**conv7-32** **conv7-32**	**conv7-32** **conv3-32**	**conv7-32** **conv3-32**	**conv7-32+** **conv3-32+**	**conv7-32** **conv3-32**	**conv7-32** **conv3-32**	**conv7-32** **conv3-32**	conv7-32+ conv3-32+
maxpool					no maxpool	maxpool		
conv3-64 **conv3-64**	**conv7-64** **conv7-64**	**conv3-64** **conv3-64**	**conv3-64** **conv3-64**	**conv3-64+** **conv3-64+**	**conv3-64** **conv3-64**	**conv3-64** **conv3-64**	**conv3-64** **conv3-64**	conv3-64+ conv3-64+
maxpool					no maxpool	maxpool		
		conv3-128 **conv3-128**	**conv3-128** **conv3-128**	**conv3-128+** **conv3-128+**	**conv3-128** **conv3-128**	**conv3-128** **conv3-128**	**conv3-128** **conv3-128**	conv3-128+ conv3-128+
conv3-64*	conv7-64*	conv3-128*	conv3-128*	conv3-128*	conv3-128*	conv3-128*	conv3-128*	conv3-128*
maxpool								
FC-128*								
FC-10*								
soft-max								

7.1 Architecture

ConvNets are well-known robust approaches for extracting features from images. Our method generates relevant, multichannel patches from a given dataset. We propose to evaluate the performance of the generated filters using a convolutional network. The VGG architecture was used as the reference network for all the experiments performed. VGG was presented in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2014 [50]. This architecture increases convolutional layers' depth sequentially and applies pooling after a certain number of layers, whereby VGG16 and VGG19 are well known. Many modifications were applied to the reference model throughout our experiments to reach better behavior with our method. Figure 24 shows a basic workflow of our method in VGG architecture.

The tested framework is a ConvNet with layers that follow a VGG block convention. In a VGG block, two or more convolutional layers are stacked with fixed padding that keeps the input dimensions after the convolution [50]. Also, we can add or not an activation function after each convolution. Besides, a max-pooling with a stride of 2 reduces these dimensions to half. Thus, this model adds these blocks one after another, reducing the dimensionality slowly and extracting features in the process. Finally, the results are flattened to a single dimension vector and fed into two fully connected dense layers to classify the input into the corresponding classes. Based on this architecture, we used a simplified VGG at most seven convolutional layers. We performed experiments to evaluate the accuracy achieved by the generated filters. Besides, we include experiments for kernel initialization using the CovTen method.

7.1.1 Configuration

The ConvNets configurations evaluated in this paper are outlined in Table 4 where the column names match their respective experiment, and the bold letters represent frozen

layers. We performed experiments in order to select parameters for our network. Then, classification experiments tested two hypotheses on our method's performance, such as the inference and the kernel initialization capacity. Some generalities across all the experiments are

1. The main experiments were executed using the CIFAR 10 dataset with 50000 training and 10000 validation images. Nevertheless, the kernels were generated with fewer samples. Other datasets were tested too and included in Supplementary Material.

2. The networks receive normalized RGB images as input, where its interval is [0-1].

3. We trained the convolution network with batches equal to 32, Adam optimizer, and a learning rate of 0.001.

4. In most cases, the activation function of each convolutional layer was ReLu, and only the C experiment replaces these functions. Indeed, C architecture changes the activation and max-pooling during the generation of covariance kernels.

7.1.2 Implementation details

The experiments were executed using the Tensorflow 2 framework. This version provides many tools, such as Keras API, to simplify all the experiments. The training of the CustomConvNets was performed on a Tesla V100 GPU with 16 GB of RAM. The training took between 30 minutes and 1 hour per experiment. Besides, source code to generate the kernels (our method) was programmed in C++, and then it was compiled as a library for its use in Tensorflow. This workflow is intended to accelerate kernel generation. The source code is available at `https://gitlab.com/ricciclope/tensor.git`.

7.2 Hyperparameters Selection

The following experiments intend to figure out the best setting for the CovTen method in a ConvNet by adding some variants.

7.2.1 Kernel Size

We tried to choose the right kernel size for our method in a ConvNet. To do that, we implemented two architectures, A.1 and A.2 (see Table 4). They were tested on the CIFAR 10 database, and the results are showed in Figure 28. The observed accuracy indicates a remarkable explicit difference between the kernel size 7×7 and 3×3 after training and validation. The highest accuracy and faster decay of the loss function came from the kernels 3×3, getting the highest performance around 80 epochs (the overfitting point). These results were expected to the findings of Simonyan and Zisserman [50], who denotes the importance of choosing small kernels over big ones in VGG models.

Nevertheless, we decided to use a configuration of kernels 7×7 in the first layer and 3×3 in the other layers. During training, we noted CovTen kernels in the first layer represent the color space and other high-level features accurately. In contrast, the deepest layers extract more complex and abstract features. Thus, this architecture proposal aims to extract all possible information on the first layer and specialized features in the next ones (see Figure 33). This setting convention was used in the following experiments.

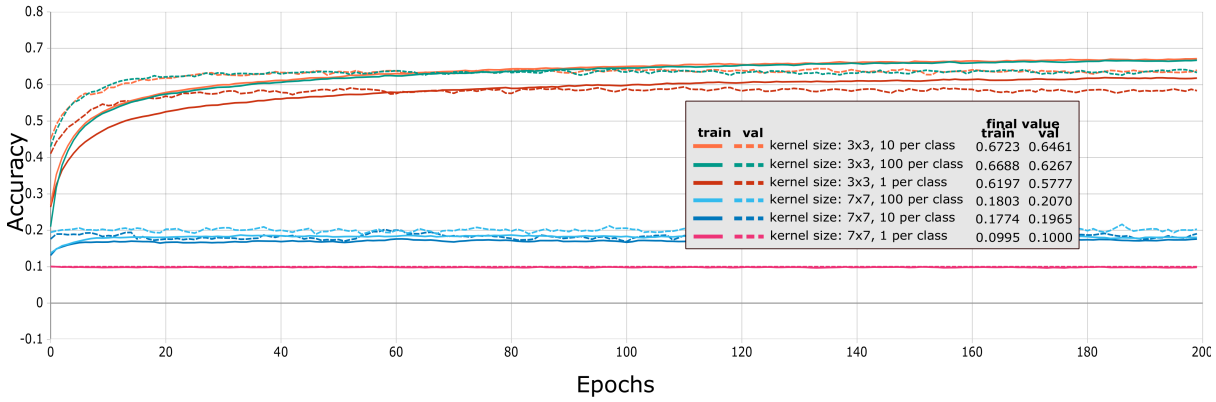


Figure 28: Kernel Size: experiment results using different kernel sizes 3×3 vs kernels 7×7, and different sub dataset sizes 1, 10, 100 images per class. The experiments were executed over CIFAR 10 dataset. The continuous line indicates the training behavior, whereas the dashed line shows the validation

7.2.2 Training Samples Number

We noted that the kernels generation with the CovTen method achieved a suitable performance with small sub-datasets and a homogeneous number of images per class. This approach was an expected result similar to the findings of Yifeng Wang et al. [62], who takes the data in the same way. Thus, this experiment intended to measure the performance with different data subsets when using our method to generate the kernels. This experiment implements architecture B in Table 4 and the CIFAR10 dataset. In this manner, Figure 29 shows the comparison between different data subsets, such as 1, 10, 100, 200, and 300 samples per class, covering the entire spectrum of characteristics of all the classes. In these results, we observe that the best accuracy and loss of training and validation data is the variant with 200 samples per class or 2000 for CIFAR 10. However, the other versions do not represent substantial changes, only around 1% to 2%. The results with 100, 200, and 300 samples per class seem to converge to equal behavior, which indicates a kind of correlation. Furthermore, we studied this possible similarity checking the correlation between kernels of the different variants. Figure 30 shows a matrix correlation example according to the number of samples per class. The results state that the correlation is low with the first sample sizes, but this keeps values close to 1 with larger sizes. It is a clear indicator that as we increase the samples per class, the filter has fewer changes. Thus, in further experiments, we decided to use a CovTen method with few samples per class.

7.2.3 Activation and Max Pooling

In the following experiment, we aim to measure the impact of activation function and max-pooling layers to generate the kernels. In this way, Figure 31 shows a comparison of combinations between no activation, leaky ReLU, and max pooling. In practice, we also tested our method with ReLU, but it returned underperforming results. Then, only the more relevant results were shown. We use the architectures C.1, C.2 and C.3 denoted in Table 4 for training each variant. Finally, we decided to follow an approach with max-pooling and no activation as our best variant so far because this model obtained the highest performance.

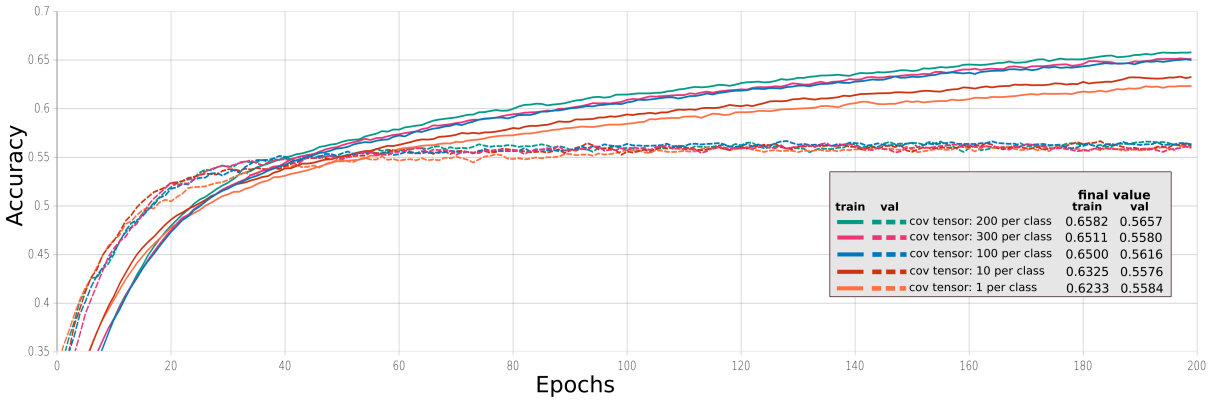


Figure 29: Training Samples Number: experiment results with different amounts of data per class (1, 10, 100, 200, 300 per class) using the CIFAR 10 dataset. The continuous line indicates the training behavior, whereas the dashed line shows the validation.

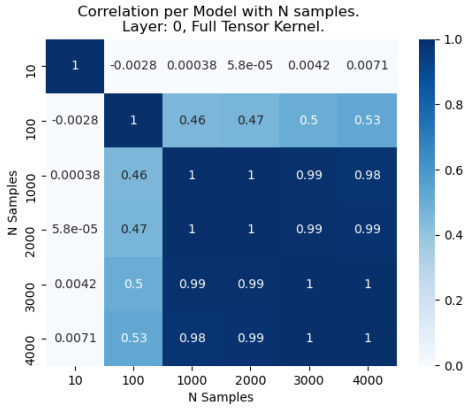


Figure 30: Correlation matrix example for a convolutional layer with the CovTen method using different samples number. Each square indicates the correlation between two experiments with its respective samples number.

7.2.4 Kernel and Feature Maps

In this section, we use our best variant so far detailed in architecture D in Table 4. In this way, Figure 32 shows some generated kernels per layer from the first to the sixth convolutional layer in each row. We can see that the first convolutional layer's kernels represent the color space and other high-level features. As we go down in the rows, we can observe that the filters extract specialized features. Besides, it is important to notice that the first row represents kernels with three channels each. Then, the next rows represent channels of just one kernel.

Figure 33 shows how the kernels filter the input in each layer, extracting different feature maps as we advance in the rows. The feature maps are arranged to describe a convolutional layer per row, beginning at the top with the first layer. Both kernels and feature maps are just a few examples of all the kernels that composed the network.

To test the feasibility of initializing weights in a neural network, we checked our method's weight distribution at epoch 0 (see Figure 34). A comparison with other initialization methods for kernels values also was added. These results show certain similarities to standard initializer methods. As we can see, kernels and features maps indicate that our method extracts features from images acceptably. For all these reasons, it seems

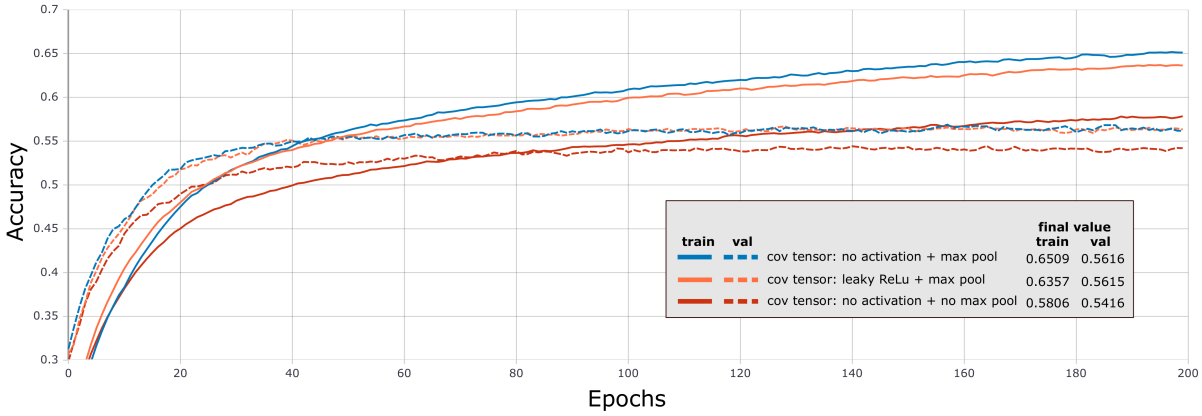


Figure 31: Activation and Max Pooling: experiment results with different setup combinations between no activaton, leaky ReLu, and max pooling using CIFAR 10 dataset. The continuous line indicates the training behavior, whereas the dashed line shows the validation.

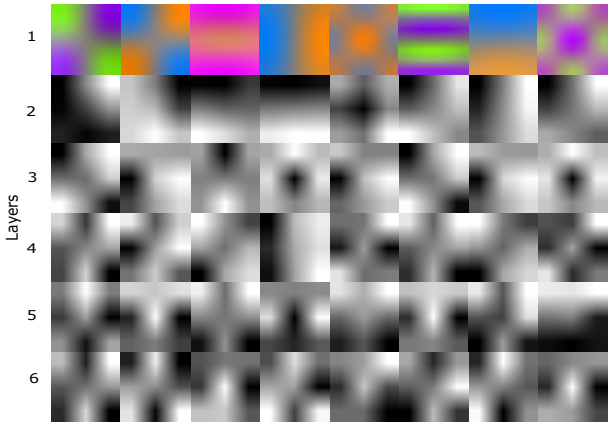


Figure 32: Kernels and Feature Maps: kernels produced by the CovTen method. They are arranged descendingly by the convolutional layer number. Elements in first row represent filters with 3 channels while elements in the following rows are channels of a single filter.

reasonable to test our method's effectiveness as a kernels initializer (see Section 7.3.2).

7.3 Classification Experiments

After different experiments, many results were obtained and analyzed to conclude the best setup. In the following section, we took the best variant to study two approaches to our method, such as Inference Capacity and Kernel Initialization.

7.3.1 Inference Capacity

As shown in previous experiments, the covariance method has an intrinsic capacity to extract the dataset's features in a single step-forward. In each experiment, we generated kernels using CovTen, and then the corresponding layers were frozen. In practical terms, our method only needs one step-forward to compute filters. A fair comparison would be to evaluate the performance versus traditional initializers at epoch 0, see Figure 35.

After filter generation, the fully connected layers were trained with the complete dataset for class inference. Because layers were frozen from the feature extraction section, this experiment aims to show inference capacity provided by the generated filters.

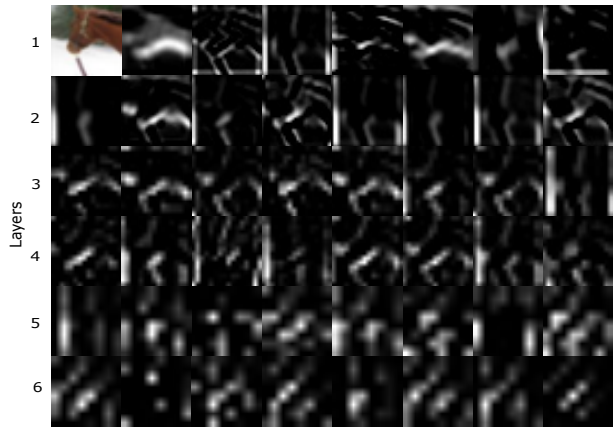


Figure 33: Kernels and Feature Maps: feature maps generated by the CovTen method using one CIFAR10 sample. They are arranged descendingly by the convolution layer number. We have to remark that the top-left element is the original image, and the other elements are channels of the corresponding feature map.

In this way, the conventional kernel initializers evaluated in this experiment were: He uniform, He normal, Glorot normal, and random normal. The different models were tested under the same conditions and architecture A (see Table 4). We also tested our method with 1, 10, 100 images per class. Note that in architecture A. in Table 4 we maintained a non-frozen ConvLayer before the fully connected step. In practice, we note that this decision increases the performance of our network.

Although our method does not update the convolutional weights, its training stage achieves to extract enough features to classify the CIFAR 10 dataset with an accuracy of around 67% (see Figure 35). Our method demonstrates an inherent and reasonable discriminative capacity because it generates kernels and uses them to classify images employing frozen-based architecture. Besides, we deployed the same experiment with CIFAR 100 (64%) and MNIST (98%) dataset, and the results showed similar behaviors (see Supplementary Material 7.6).

7.3.2 Covariance Method as Kernel Initializer

The kernel initialization is one of the main setups that a network architecture needs to perform suitable training. The CovTen method has demonstrated a sufficient feature extraction capacity, which can be used in several ways. Thus, in Figure 36, we took advantage of this quality to initialize the kernel weights and train the network. For this experiment, The models were trained under similar setups, and the unique variant was the kernel initialization. Our model was compared with well-known kernel initializer methods such as uniform, normal, Glorot normal, and random normal. Besides, we evaluated the covariance method with different samples number: 1, 10, 100 per class.

The results show that our method is comparable and proportional to conventional kernel initializer methods using the CIFAR 10 dataset, returning an accuracy of around 91%. The covariance method got close results to the traditional methods in training as well as validation. Moreover, we tested the same experiment with other datasets such as CIFAR100 (72%) and MNIST (99%) (See Supplementary Material 7.7). These results showed similar behavior, and, just with MNIST, we got a light improvement of accuracy with our method.

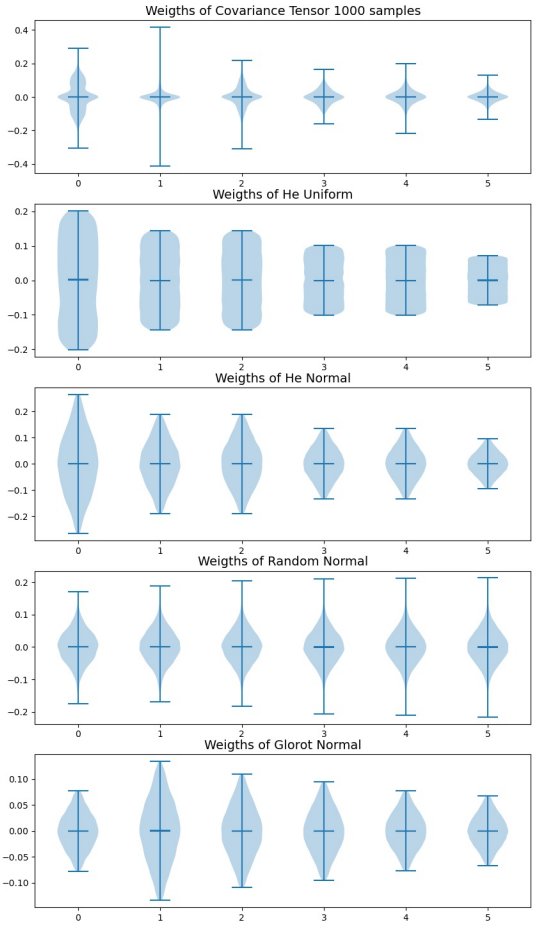


Figure 34: CovTen as Kernel Initializer: violin plots of weight distribution in each convolution layer at epoch zero, using architecture B of classification experiments. Different weights initialization methods are plotted along with the results of the CovTen method.

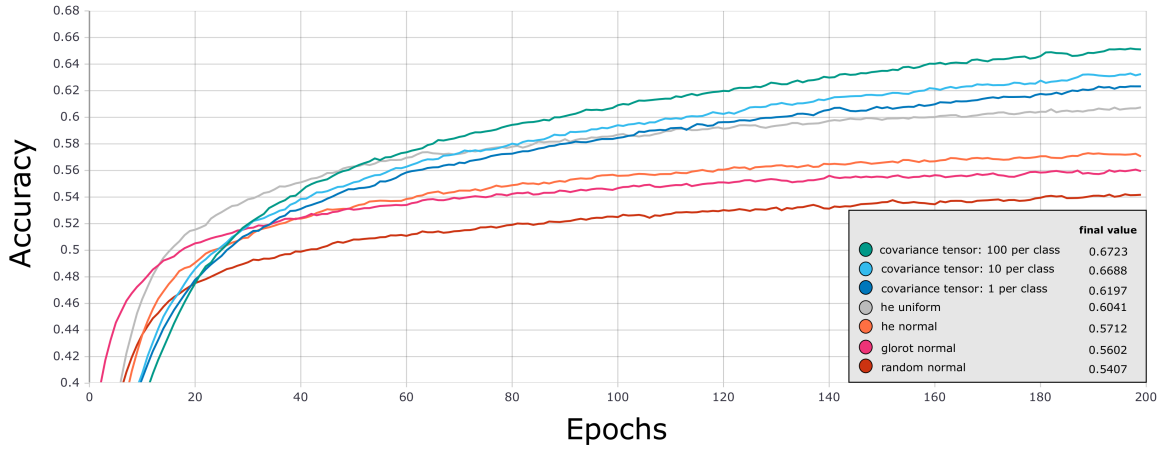


Figure 35: Inference Capacity: experiment results with different kernel initializers and samples per class, used in a model with frozen layers. The experiments were executed over CIFAR 10 dataset.

7.4 Architecture Comparison

In this section, we aim to compare our method with state-of-the-art methods. We review the literature to check common architectures tested under the CIFAR 10 dataset. Table

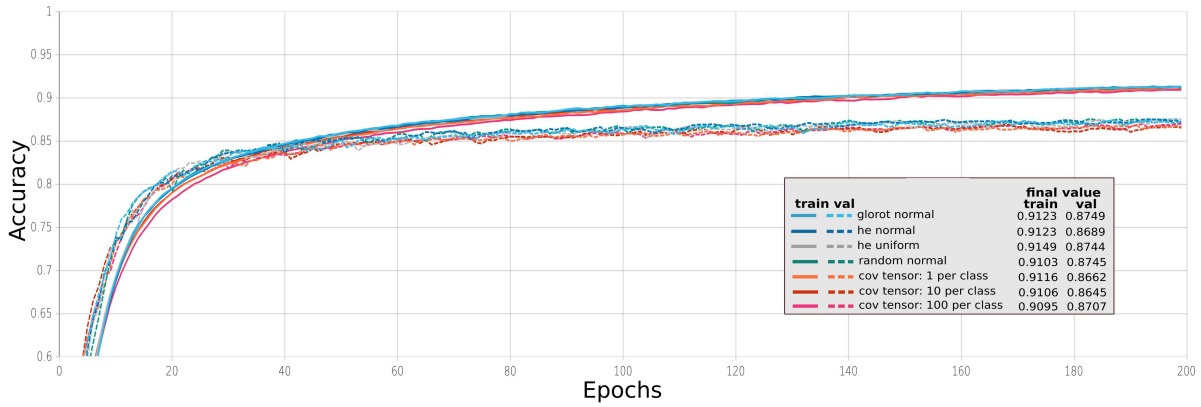


Figure 36: CovTen Method as Kernel Initializer: experiment results with different kernel initializer methods and different samples per class, performing a full training over the complete network (without frozen model). The experiments were executed over CIFAR 10 dataset.

5 shows the accuracy, flops, and params per method.

Table 5: Comparison between different state-of-the-art methods and our method under **CIFAR 10** dataset. Our method is highlighted with a gray color.

Methods	**Accuracy (%)**	**MFLOPS**	**Params (M)**
PCA Net without pyramidal pooling (5000 samples in total)	46.21	–	–
VGG 7 + CovTensor with frozen layers (1, 10, 100 samples per class)	56.02, 56.34, 56.46	105	0.412
PCA Net [10] (50000 samples)	78.67	–	–
AlexNet-tf [48]	82	–	–
Random Features 256k-aug [12] [48]	85.6	–	–
CudaConvNet [48]	88.5	–	–
VGG 7 + CovTensor as kernel initializer (1, 10, 100 samples per class)	86.6, 86.9, 87.07	105	0.703
VGG16 [4] [37]	93.25, 93.6	313	15
VGG16 prunned [4]	93.4	206	5.4
ResNet-110 Basic [22]	93.5	–	1.7
VGG-19 with GradInit [67]	94.71	–	20.03
NAT-M1 [41]	97.4	232	4.3
MUXNet-m [40]	98	200	2.1
NAT-M4 [41]	98.4	468	6.9
PyramidNet-272 [66]	98.71	6330	32.6

The most explicit comparison is between our method and PCA Net. Table 5 shows that our method with frozen layers effectively achieves better accuracy than PCA Net without pyramidal pooling but struggles compared with the complete PCA Net. We hypothesize that the addition of spatial pyramidal pooling (SPP) is crucial to deal with the invariant poses and complex background, mainly found in CIFAR 10 dataset. We

suggest in future work include an SPP according to our method needs. Despite this accuracy drawback, we can note that our method with frozen layers achieves a promising result with fewer samples than the PCA Net and less kernel generation time. Compared to PCANet, our method uses a tensor decomposition which establishes linear dependency between dimensions and reduces the curse of dimensionality. These features allow capturing spatial information with fewer samples. Additionally, we can generalize the dependency between dimensions, i.e., height, width, and channels, to generate multichannel filters.

On the other hand, our method as kernel initializer overcomes accuracy achieved by Alex Net, Random Features, and Cuda Conv Net. However, our method accuracy underperforms compared with more recent architectures. We expected these results because we are using a simple VGG architecture with no more than seven convolutional layers, but even so, it achieves results nearly to VGG16 and ResNet 110. This fact is clear evidence of why we also get fewer flops and parameters compared with modern architectures. These results (in terms of accuracy, flops, and parameters) indicate our method's effectiveness in retaining spatial information. In future works, we suggest experimenting with other deeper and modern architectures.

Despite our results, we consider that this comparison is not entirely fair because we are not using all the capabilities we expected from our method. In general, our method presents some promising results compared with other architectures, but it needs to exploit more features to achieve its best capability. Suggested improvements are related to the number of samples during training and including a more profound (or complex) architecture.

7.5 Additional Feature Maps

Figure 37 shows some feature maps examples of other tested images, which was not included during kernel generation. The feature maps are downward arranged, starting at the top with the first layer and ending with the sixth layer.

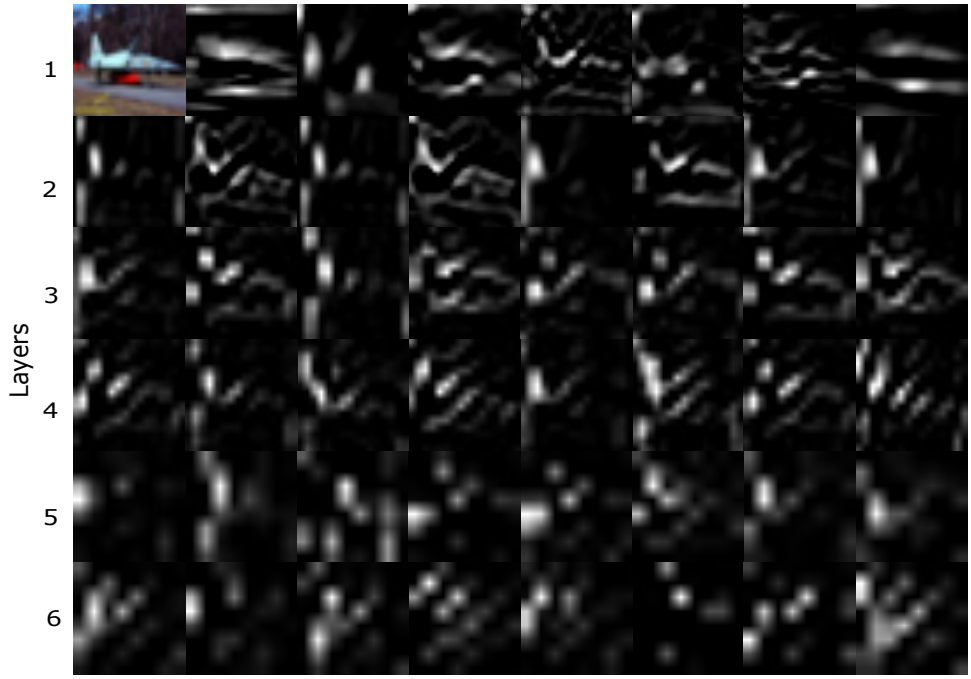


Figure 37: Kernels and Feature Maps: feature maps generated by the CovTen method using one CIFAR10 sample. They are arranged descendingly by the convolution layer number. We have to remark that the top-left element is the original image, and the other elements are channels of the corresponding feature map.

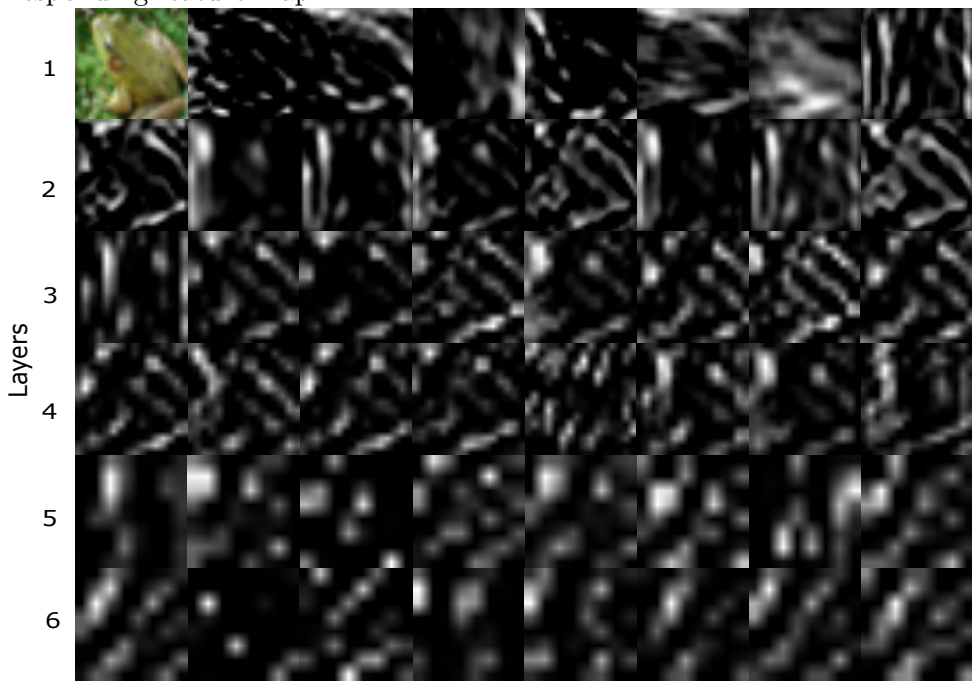


Figure 38: Kernels and Feature Maps: feature maps generated by the CovTen method using one CIFAR10 sample. They are arranged descendingly by the convolution layer number. We have to remark that the top-left element is the original image, and the other elements are channels of the corresponding feature map.

7.6 Additional Inference Capacity Experiments

7.6.1 CIFAR 100

These experiments were tested under similar conditions and the convention of freezing layers. The fully-connected layers were updated according to the needs of the dataset. Thus, in this case, we introduce shallow, dense layers to classify 100 classes (see Figure 39). In this way, the training stage achieves enough features to classify with an accuracy of around 64%.

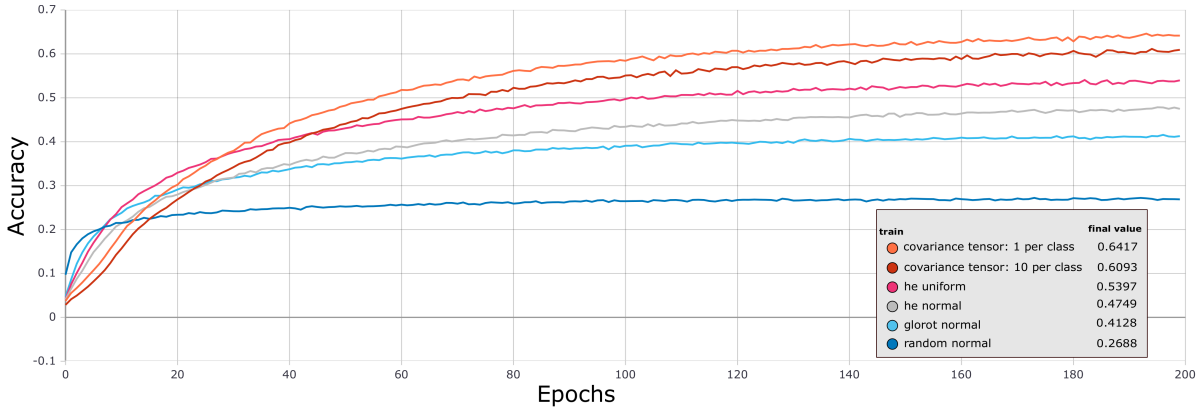


Figure 39: Inference Capacity: experiment results with different kernel initializers and samples per class, used in a a model with frozen layers. The experiments were executed over CIFAR 100 dataset.

7.6.2 MNIST

These experiments were tested under similar conditions and the convention of freezing layers. The fully-connected layers were updated according to the needs of the dataset. Thus, in this case, we introduce shallow, dense layers to classify ten classes (see Figure 39). In this way, the training stage achieves enough features to classify with an accuracy of around 98%.

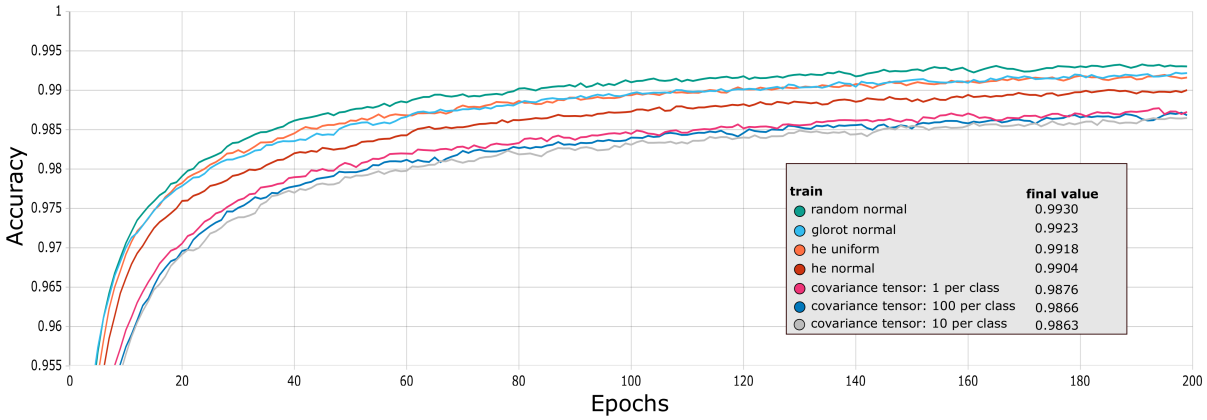


Figure 40: Inference Capacity: experiment results with different kernel initializers and samples per class, used in a model with frozen layers. The experiments were executed over the MNIST dataset.

7.7 Additional Kernel Initializers Experiments

7.7.1 CIFAR 100

These results were obtained under the same conditions. The fully-connected layers were updated according to the needs of the dataset. Thus, in this case, we introduce shallow, dense layers to classify 100 classes (see Figure 41). In this way, the training stage returns an accuracy of around 72%.

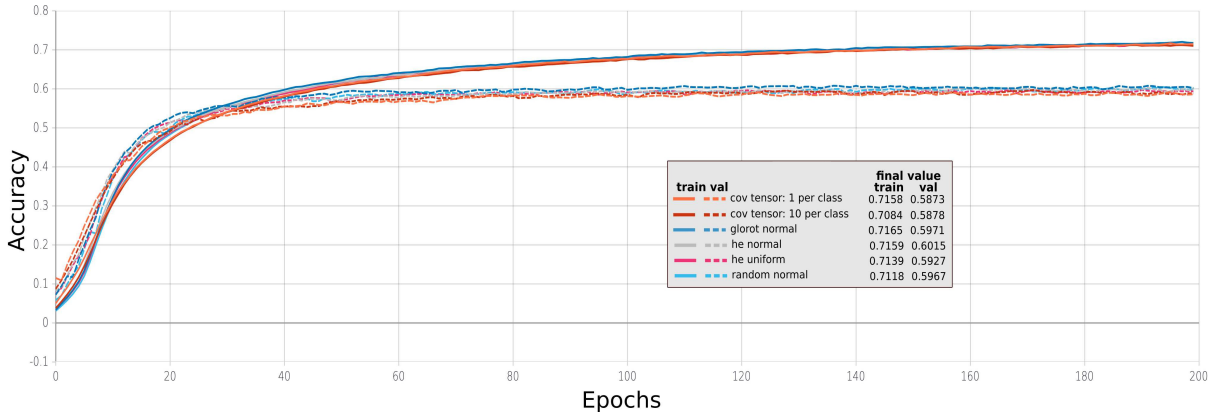


Figure 41: Covariance Method as Kernel Initializer: experiment results with different kernel initializer methods and different samples per class, performing a full training over the complete network (without frozen model). The experiments were executed over CIFAR 100 dataset.

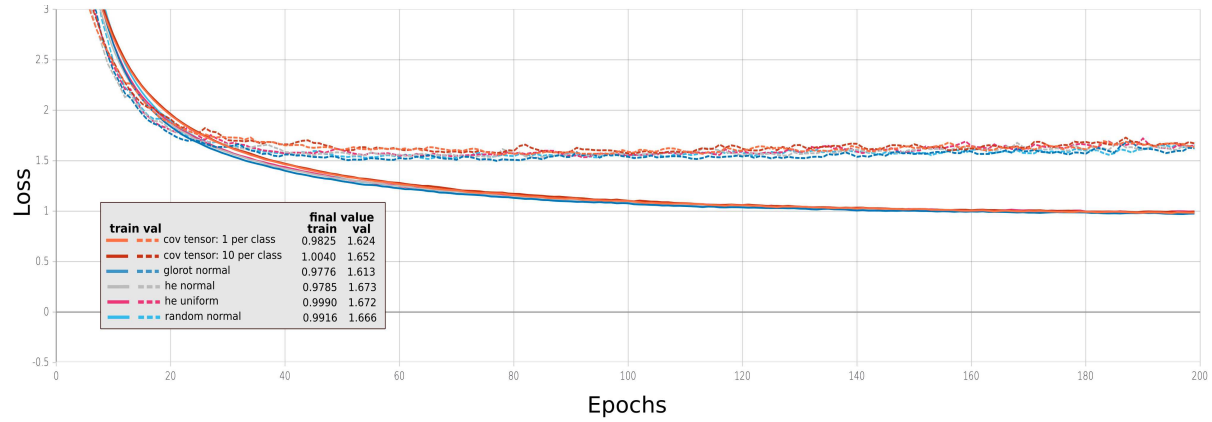


Figure 42: Covariance Method as Kernel Initializer: experiment results with different kernel initializer methods and different samples per class, performing a full training over the complete network (without frozen model). The experiments were executed over CIFAR 100 dataset.

7.7.2 MNIST

We used the same parameters for these tests. The fully-connected layers were updated according to the needs of the dataset. Thus, in this case, we introduce shallow, dense layers to classify ten classes and small input size condition 28. These experiments also present a light improvement of performance to the conventional methods (see Figure 43). In this way, the training stage returns an accuracy of around 99%.

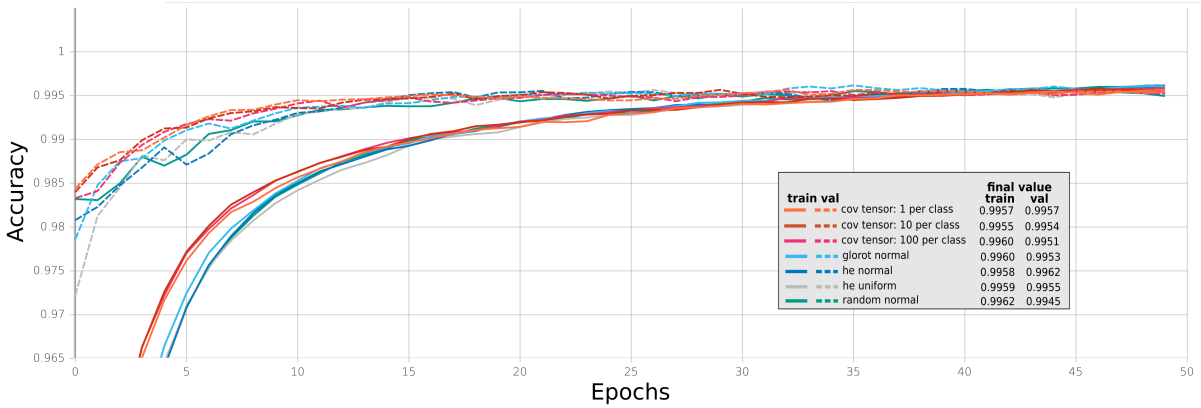


Figure 43: CovTen Method as Kernel Initializer: experiment results with different kernel initializer methods and different samples per class, performing a full training over the complete network (without frozen model). The experiments were executed over the MNIST dataset.

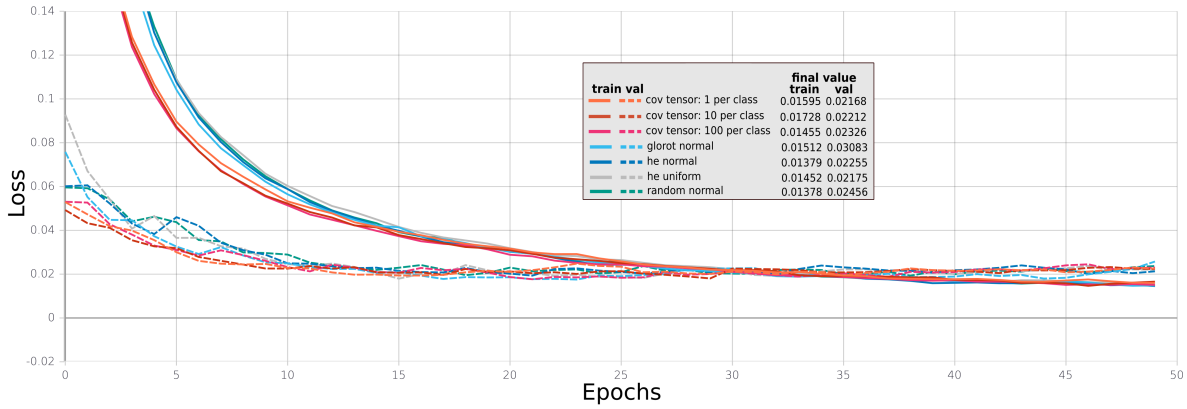


Figure 44: MNIST: train and validation loss for covariance method as kernel initializer

8 Conclusion

In this work, we have shown the feature extraction capability of the CovTen method when it is attached to a ConvNet, demonstrating that our technique learns multichannel filters from the dataset samples. We tested our method against available initializers (He random, He uniform, Glorot normal, random normal), confirming linear tensor constraints' usefulness to generate features kernels. Our method proposed establishing linear dependency between width, height, and channel dimensions by generating a covariance tensor and decomposing it with a Tucker Model. Experiments were able to confirm the reduction in samples required to generate filters. On figure 30 showed the filter correlation generated by different amounts of samples. This result showed that filters generated using above 200 samples were highly correlated, thus, indicating the number of samples was enough. This behavior strengthens the hypothesis that linear dependency between dimensions reduces the dimensional complexity. We were able to achieve better performance versus available initializers using at least 1 sample per class in CIFAR 100.

In this way, we could discriminate classes by a single feed-forward with increased accuracy over available initializers, around 67% and 64% for CIFAR10 and CIFAR100 datasets. It is important to remark that labels were not used in kernels generation. The generated filters were tested as kernel initializers and achieved similar performance to state-of-the-art, around 91% CIFAR10, 72% CIFAR100, and 99% MNIST. Finally, our method was compared with some state-of-the-art architectures to check the performance in terms of accuracy, flops, and parameter number. This comparison demonstrated the effectiveness of our method and some possible drawbacks and paths to take to avoid them.

8.1 Future Work

A promising path is to establish linear dependency constraints during back propagation training. Filter selection was performed according its norm, a different filter selection algorithm could improve performance.

The filter generation method accepts as input a third-order tensor, so it has to iterate in a loop over all the batch images. To gain access to all the images, we have to load them in memory. This loading takes high hardware capacity limiting our method to the memory size. However, this method can be enhanced by changing the function's input to a fourth-order tensor with the batch as a fourth dimension. The advantage of using the fourth dimension in the input is that data does not have to be loaded directly in memory. In contrast, we can use a place holder for it and map our function aside from all the other operations in the ConvLayer. A small change that can take advantage of all the optimizations currently available in machine learning frameworks improves computation time.

Another aspect to include for future work is our method's inclusion in other well-known state-of-the-art architectures. With this addition, we can compare the CovTen method's performance across frameworks. Besides, it is possible to analyze the extracted features as the network increases its depth, as in our experiments, where the first layers extract more heterogeneous features than the deeper layers. After that, we could better understand our method's performance and develop a specific architecture that takes advantage of our tested features.

9 Appendix

9.1 Function of a scalar random variable

Suppose a scalar random variable $X \sim f_X(x)$, and define an invertible and differentiable transformation $g(X) = Y : \mathbb{R} \to \mathbb{R}$. The distribution of the transformed random variable $Y \sim f_Y(y)$ can be computed using the change of variable method. Derivation start with the definition of cumulative distribution function for Y as stated on equation 1.

$$F_Y(y) = P(Y \leq y) = P(g(X) \leq y) \tag{85}$$

At this point the relation to be made is to compare the cumulative distributions of Y and X. The idea behind this comparison is the cumulative probability $F_Y(y)$ at a point y should be the same that the cumulative probability $F_X(g^{-1}(y))$ at a point $g^{-1}(y)$. In fact the function of a random variable modifies the pdf domain e.g. the horizontal axis when looking at a pdf graph. On figure 3 is depicted the transformation of a normal random variable given by $Y = X/2$. The parameters used were $\mu = 0$, $\sigma^2 = 4$. Figure 3 depicts the change of shape due to the transformation, furthermore show the cumulative distributions compared. In particular this transformation shrinks the random variable space.

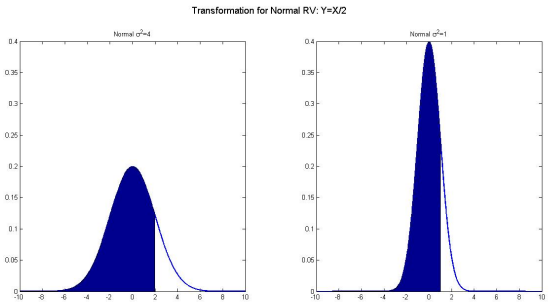


Figure 45: Transformation of a random variable.

The comparison of the cumulative distributions are stated at equation 2.

$$\begin{aligned} F_Y(y) &= P(Y \leq y) = P(g(X) \leq y) \\ &=^{(a)} P(X \leq g^{-1}(y)) \\ &= \int_{-\infty}^{g^{-1}(y)} f_X(x)dx \\ &=^{(b)} \int_{-\infty}^{y} f_X(g^{-1}(y)) \left| \frac{dg^{-1}(y)}{dy} \right| dy \end{aligned} \tag{86}$$

On (a) its considered a growing function $g(x)$, in the case we have a decreasing one the inequality change its side and the probability complement should be used. On (b) the change of variable is performed using the differential $dx = \left| \frac{dg^{-1}(y)}{dy} \right| dy$. The absolute value appears do to the fact $g^{-1}(y)$ could be growing or decaying but we are interested on its rate change rather than its direction.

By definition the pdf is the derivative of the cdf. Finally the probability distribution function of the transformation of a scalar random variable is stated in equation 4.

$$
\begin{aligned}
f_Y(y) = F'_Y(u) &= \frac{d}{dy}\int_{-\infty}^{y} f_X(g^{-1}(y)) \left|\frac{dg^{-1}(y)}{dy}\right| dy \\
&= f_X(g^{-1}(y)) \left|\frac{dg^{-1}(y)}{dy}\right|
\end{aligned}
\tag{87}
$$

9.2 Function of a vector random variable

Let $S \in \mathbb{R}^n$ be a vector random variable and $g(S) = Ax$, $g(S) : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible transformation. We are interested on the "joint" distribution of the vector random variable $X \in \mathbb{R}^n$. The joint distribution of vector random variable is $S \sim f_S(s)$, and assume each component s_i is independent of the others, i.e. $P_S(s_1, \ldots, s_n) = \prod_i^n P_{S_i}(s_i)$.

We used the generalization of the change of variables technique for vector random variables. We followed the same procedure as in the previous section for scalar random variable. Let's start with the definition of cumulative distribution for X and compare to the cumulative of S. This process is stated at equation 4.

$$
\begin{aligned}
F_X(x) &= P_X(X \leq x) = P_X(As \leq x) \\
&= P_S(s \leq A^{-1}x) \\
&= \int_{-\infty}^{A^{-1}x} f_S(s)ds \\
&=^{(a)} \int_{-\infty}^{x} f_S(A^{-1}x) \left|det\left(J(A^{-1}x)\right)\right| dx
\end{aligned}
\tag{88}
$$

On (a) the change of variable performed was given by the differential of $s = A^{-1}x$, i.e. $ds = |det\,(J(A^{-1}x))|\,dx$. Where $J(\dot{})$ is the Jacobian matrix of all first order partial derivatives of the vector valued function $A^{-1}x$. Using the definition of cumulative distribution, we derived the joint distribution of the vector random variable X in equation 5.

$$
f_X(x) = F'_X(x) = f_S(A^{-1}x) \left|det\left(J(A^{-1}x)\right)\right| \tag{89}
$$

It's important to state the change of variable for vector variables include the determinant of a Jacobian, this term represents the change of volume produced by the transformation. The absolute value on the determinant indicates we are interested only on the change rate rather than the direction.

Note that if you want to compute the distribution of a single X_i instead of the joint distribution, the path is to use the definition of marginal distribution.

9.3 Tensor by matrix product (Mode n multiplication)

Beyond the n-mode multiplication proposed on [15], [13], [31], we used a n-frame linear combination approach to perform tensor-matrix product. Each n-mode product can be understood as a linear combination of n-frames.

There are two steps to perform the n-frame product $\underline{X} \times_n U^{(n)}$; first identify the frame of $\underline{X}$ on the corresponding mode (figure 47), then perform linear combination of the frames according to a row of the multiplying matrix $U^{(n)}$. Each row of the multiplying matrix create a new n-frame.

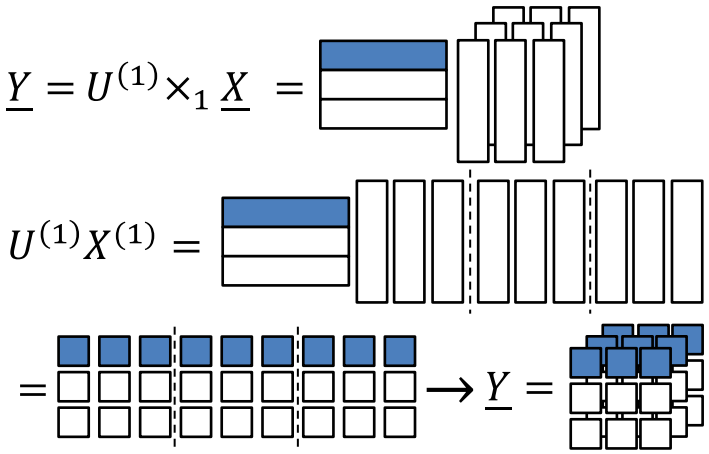


Figure 46: Mode 1 product between a third order tensor and a matrix using unfolding algorithm.

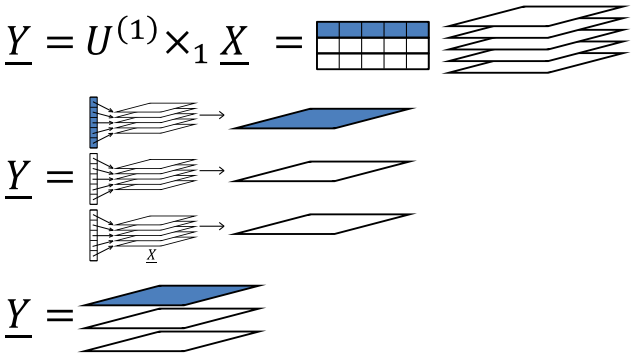


Figure 47: Mode 1 product between a third order tensor and a matrix using the linear combination of frames algorithm.

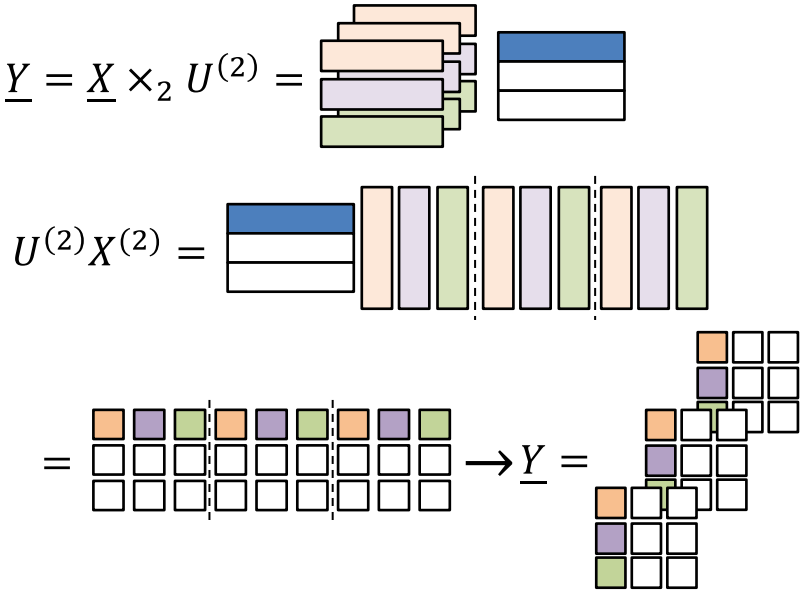


Figure 48: Mode 2 product between a third order tensor and a matrix using unfolding algorithm.

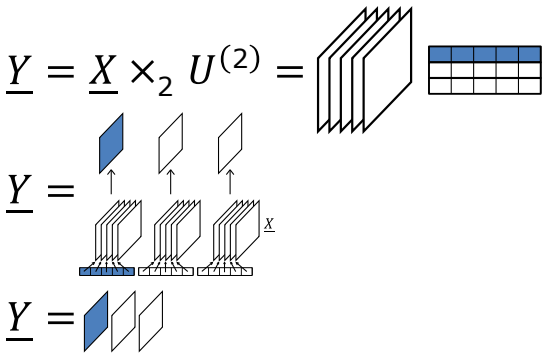


Figure 49: Mode 2 product between a third order tensor and a matrix using the linear combination of frames algorithm. The case of a matrix multiplied on mode 2 by other matrix, we have $X \times_2 V = XV^T$

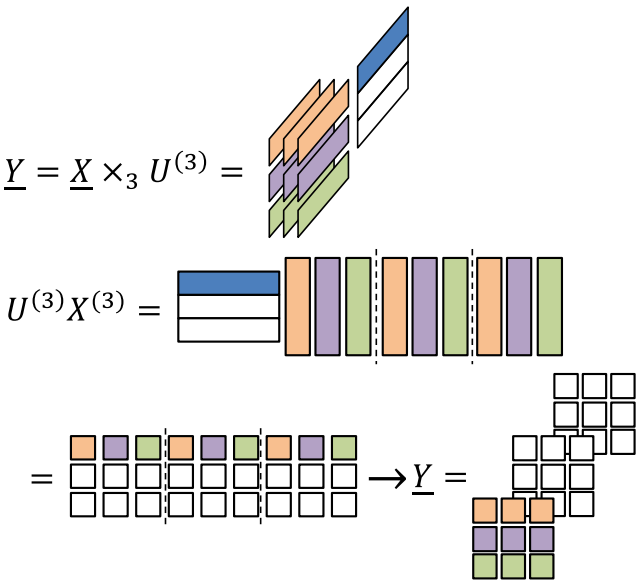


Figure 50: Mode 3 product between a third order tensor and a matrix using unfolding algorithm.

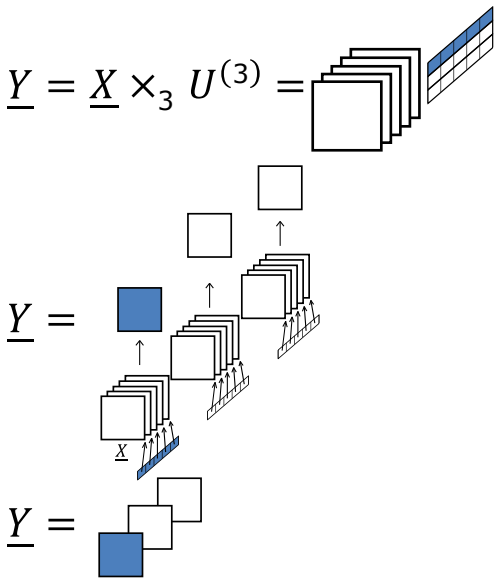


Figure 51: Mode 3 product between a third order tensor and a matrix using the linear combination of frames algorithm.

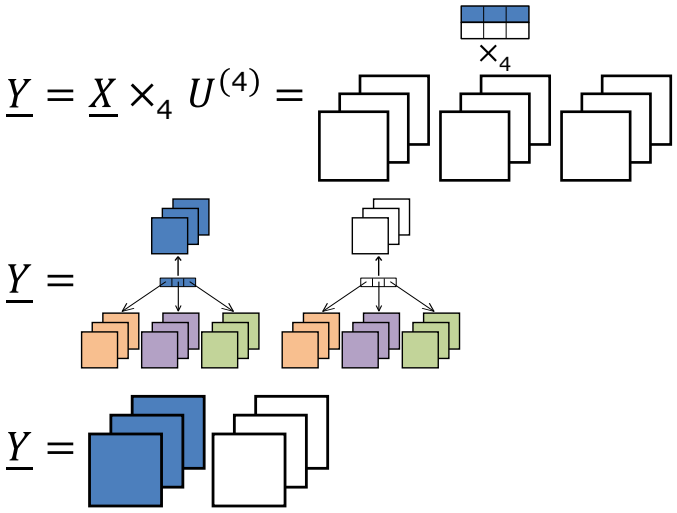


Figure 52: Mode 4 product between a fourth order tensor and a matrix using the linear combination of cubes algorithm.

10 Bibliography

References

[1] E. Acar, C. A. Bingol, H. Bingol, R. Bro, and B. Yener. Seizure recognition on epilepsy feature tensor. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 4273–4276. IEEE, 2007.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

[3] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. Tthresh: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics*, 2019.

[4] F. Bordes, S. Honari, and P. Vincent. Learning to generate samples from noise through infusion training. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[5] R. Bro. Multiway calibration. multilinear pls. *Journal of chemometrics*, 10(1):47–61, 1996.

[6] R. Bro. Review on multiway analysis in chemistry—2000–2005. *Critical reviews in analytical chemistry*, 36(3-4):279–293, 2006.

[7] R. Bro, A. K. Smilde, and S. de Jong. On the difference between low-rank and subspace approximation: improved model for multi-linear pls regression. *Chemometrics and Intelligent Laboratory Systems*, 58(1):3–13, 2001.

[8] J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

[9] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970.

[10] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification? *IEEE transactions on image processing*, 24(12):5017–5032, 2015.

[11] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, 2015.

[12] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[13] L. De Lathauwer. *Signal processing based on multilinear algebra*. Katholieke Universiteit Leuven, 1997.

[14] L. De Lathauwer. Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–1066, 2008.

[15] 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.

[16] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[17] R. Fonseca, O. Guarnizo, D. Suntaxi, A. Cadiz, and W. Creixell. Convolutional neural network feature extraction using covariance tensor decomposition. *IEEE Access*, 9:66646–66660, 2021.

[18] Y. Fu and T. S. Huang. Image classification using correlation tensor analysis. *IEEE Transactions on Image Processing*, 17(2):226–234, 2008.

[19] W. Guo, I. Kotsia, and I. Patras. Tensor learning for regression. *IEEE Transactions on Image Processing*, 21(2):816–827, 2011.

[20] R. A. Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 1970.

[21] K. Hasegawa, M. Arakawa, and K. Funatsu. Rational choice of bioactive conformations through use of conformation analysis and 3-way partial least squares modeling. *Chemometrics and Intelligent Laboratory Systems*, 50(2):253–261, 2000.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[24] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[25] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[26] Jian Yang, D. Zhang, A. F. Frangi, and Jing-yu Yang. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):131–137, 2004.

[27] A. Kembhavi, D. Harwood, and L. S. Davis. Vehicle detection using partial least squares. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1250–1265, 2011.

[28] M. Kirby and L. Sirovich. Application of the karhunen-loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern analysis and Machine intelligence*, 12(1):103–108, 1990.

[29] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe. The tensor algebra compiler. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–29, 2017.

[30] T. G. Kolda. *Multilinear operators for higher-order decompositions*. United States. Department of Energy, 2006.

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

[32] J. Kossaifi, Z. C. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar. Tensor regression networks. *Journal of Machine Learning Research*, 21:1–21, 2020.

[33] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic. Tensorly: Tensor learning in python. *The Journal of Machine Learning Research*, 20(1):925–930, 2019.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[35] R. Leardi. Multi-way analysis with applications in the chemical sciences, age smilde, rasmus bro and paul geladi, wiley, chichester, 2004, isbn 0-471-98691-7, 381 pp. *Journal of Chemometrics*, 19(2):119–120, 2005.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*, pages 730–734. IEEE, 2015.

[38] H. Lu, K. N. Plataniotis, and A. Venetsanopoulos. *Multilinear Subspace Learning: Dimensionality Reduction of Multidimensional Data*. CRC press, 2013.

[39] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. Multilinear principal component analysis of tensor objects for recognition. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 776–779. IEEE, 2006.

[40] Z. Lu, K. Deb, and V. N. Boddeti. Muxconv: Information multiplexing in convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12044–12053, 2020.

[41] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti. Neural architecture transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[42] A. Mandal and A. Cichocki. Non-linear canonical correlation analysis using alpha-beta divergence. *Entropy*, 15(7):2788–2804, 2013.

[43] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[44] E. Martınez-Montes, P. A. Valdés-Sosa, F. Miwakeichi, R. I. Goldman, and M. S. Cohen. Concurrent eeg/fmri analysis by multiway partial least squares. *NeuroImage*, 22(3):1023–1034, 2004.

[45] J. Nilsson, S. d. Jong, A. K. Smilde, et al. Multiway calibration in 3d qsar. *Journal of chemometrics*, 11(6):511–524, 1997.

[46] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[47] M. A. Raihan, N. Goli, and T. M. Aamodt. Modeling deep learning accelerator enabled gpus. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 79–92. IEEE, 2019.

[48] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

[49] L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1233–1240, 2013.

[50] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] A. K. Smilde. Comments on multilinear pls. *Journal of Chemometrics*, 11(5):367–377, 1997.

[52] A. K. Smilde, H. A. Kiers, et al. Multiway covariates regression models. *Journal of Chemometrics*, 13(1):31–48, 1999.

[53] L. Song, B. Du, L. Zhang, L. Zhang, J. Wu, and X. Li. Nonlocal patch based t-svd for image inpainting: Algorithm and error analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[54] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[55] S. Theodoridis and K. Koutroumbas. Chapter 6 - feature generation i. In S. Theodoridis and K. Koutroumbas, editors, *Pattern Recognition (Fourth Edition)*, pages 91–150. Academic Press, Boston, fourth edition edition, 2009.

[56] L. R. Tucker. The extension of factor analysis to three-dimensional matrices. *Contributions to mathematical psychology*, 110119, 1964.

[57] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

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

[59] M. A. O. Vasilescu and D. Terzopoulos. Multilinear image analysis for facial recognition. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 511–514. IEEE, 2002.

[60] M. A. O. Vasilescu and D. Terzopoulos. Multilinear subspace analysis of image ensembles. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–93. IEEE, 2003.

[61] S. Verma, W. Liu, C. Wang, and L. Zhu. Hybrid networks: Improving deep learning networks via integrating two views of images. In *International Conference on Neural Information Processing*, pages 46–58. Springer, 2018.

[62] Y. Wang, Y. Rong, H. Pan, K. Liu, Y. Hu, F. Wu, W. Peng, X. Xue, and J. Chen. Pca based kernel initialization for convolutional neural networks. In Y. Tan, Y. Shi, and M. Tuba, editors, *Data Mining and Big Data*, pages 71–82, Singapore, 2020. Springer Singapore.

[63] J. Wu, S. Qiu, R. Zeng, Y. Kong, L. Senhadji, and H. Shu. Multilinear principal component analysis network for tensor object classification. *IEEE Access*, 5:3322–3331, 2017.

[64] L. Zhang, L. Song, B. Du, and Y. Zhang. Nonlocal low-rank tensor completion for visual data. *IEEE transactions on cybernetics*, 2019.

[65] Q. Zhao, C. F. Caiafa, D. P. Mandic, Z. C. Chao, Y. Nagasaka, N. Fujii, L. Zhang, and A. Cichocki. Higher order partial least squares (hopls): a generalized multilinear regression method. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1660–1673, 2013.

[66] S. Zhao, L. Zhou, W. Wang, D. Cai, T. L. Lam, and Y. Xu. Splitnet: Divide and co-training. *arXiv preprint arXiv:2011.14660*, 2020.

[67] C. Zhu, R. Ni, Z. Xu, K. Kong, W. R. Huang, and T. Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *arXiv preprint arXiv:2102.08098*, 2021.

[68] K. D. Zissis, R. G. Brereton, S. Dunkerley, and R. E. Escott. Two-way, unfolded three-way and three-mode partial least squares calibration of diode array hplc chromatograms for the quantitation of low-level pharmaceutical impurities. *Analytica chimica acta*, 384(1):71–81, 1999.