Dictionary Learning for data compression within a Digital Twin Framework

Laura Cavalli[1,*], Domitilla Brandoni[1], Margherita Porcelli[2,3] and Eric Pascolo[1]

[1]CINECA, Via Magnanelli 2, Casalecchio di Reno (BO), 40033, Italy
[2]Dipartimento di Ingegneria Industriale, Università degli Studi di Firenze, Viale Morgagni 40/44, 50134, Firenze, Italy
[3]ISTI–CNR, Via Moruzzi 1, Pisa, Italy. INdAM Research Group GNCS.

Abstract
Digital Twin system plays a crucial role in several contexts, from smart agriculture to predictive maintenance, from healthcare to weather modelling. To be effective, it involves a continuous exchange of massive data between IoT sensors on real world and digital system hosted on HPC and vice versa. Nevertheless, the transmitted signals often exhibit high similarity, resulting in a redundant dataset very suitable for compression. This paper shows how Dictionary Learning can be used as a preprocessing technique for AI algorithms due to its ability to compress large data volumes up to 80% with a potential enhancement of the performances acting both as a denoising and compression technique. This algorithm operates efficiently on various types of datasets, from images to timeseries, and is well-suited for deployment on devices with limited computational resources, like IoT sensors.

Keywords
Digital Twin, Dictionary Learning, parallel OMP, timeseries compression, images compression, anomaly detection, image recognition

1. Introduction

A digital twin can be simply seen as a system consisting of two entities, a tangible, subject-of-interest, and its digital replica, interconnected by a continuous stream of data. In this context, data reflecting the physical entity are acquired through IoT sensors and sent to a dedicated HPC which constitutes its digital mirror. Within the HPC, data undergoes AI analysis to simulate the behavior and potential scenarios of the physical entity. The resulting insights are looped back into the physical system, impacting decision-making. An efficient transmission and storage of such large volumes of sensor data are therefore crucial to reduce latency between the two systems ensuring a reliable real-time digital representation, but this is often prohibitively expensive. For this reason, it is necessary to explore compression algorithms that lighten and speed up data transmission while preserving their meaningful information. Among the available state-of-the-art compression tools, we explore Dictionary Learning (DL), a robust sparse matrix factorization approach. Given a matrix of signals Y, DL is able to learn a sparse representation $Y \approx DX$ expressing each signal as a linear combination of few basis elements, called atoms, which constitute the columns of D. In this work we will show that DL has various features that make it very suitable for use in data compression and transmission: i) it enables exceptional compression of redundant data due to its distinctive sparse factorization feature; ii) it is a versatile approach being able to handle diverse data types, including images and time series; iii) its solution can be performed with an algorithm, supplied in this work, with low computational resource demand and independent of specific libraries, making it lightweight and well-suited for edge computing.

The literature on DL comprises many applications across various fields, including denoising, inpainting, classification, and compression. Regarding data compression, an interesting online DL approach is proposed in [1] where massive datasets streamed through in a preset order are compressed and denoised. Furthermore, the work [2] presents CORAD, a novel DL-based compression algorithm for time series which is able to harness the correlation across multiple related time series to eliminate redundancy performing a more efficient compression. However, as far as we know, this work is the first to incorporate DL as a compression method within the Digital Twins (DT) domain, using it as a powerful preprocessing technique for both time series and images. Also, we developed an optimized DL algorithm for increasing its lightweight and efficiency in the DT framework.

This work is structured as follows: Section II gives a brief overview of the DL problem and of its solution. Section III integrates the DL approach within a DT framework and presents the overall DL4DT workflow, while Section IV discusses numerical results, conducting a de-

Ital-IA 2024: 4th National Conference on Artificial Intelligence, organized by CINI, May 29-30, 2024, Naples, Italy
*Corresponding author.

l.cavalli@cineca.it (L. Cavalli); d.brandoni@cineca.it (D. Brandoni); margherita.porcelli@unifi.it (M. Porcelli); e.pascolo@cineca.it (E. Pascolo)
0000-0002-8157-1459 (D. Brandoni); 0000-0003-0183-1204 (M. Porcelli)

tailed analysis of the algorithm performance across various datasets. Additionally, it introduces several techniques designed to improve the algorithm execution speed. All the codes necessary to reproduce the experiments shown in this paper are available at the following link: https://github.com/Eurocc-Italy/DL4DT.

2. Dictionary Learning overview

The aim of DL is to discover an overcomplete set of basis functions (atoms) able to represent in a sparse manner a given set of data samples. Given a matrix of training signals $Y \in \mathbb{R}^{m\times N}(m \ll N)$, DL seeks to find a dictionary $D \in \mathbb{R}^{m\times n}(m \ll n)$ and a sparse matrix $X \in \mathbb{R}^{n\times N}$ to represent $Y \approx DX$. The DL problem can be formulated in many equivalent ways, each one promoting a different aspect of the problem as shown in detail in [3]. In this case we decided to formulate it as a two variable, non-convex, constrained optimization problem of the form

$$\min_{D,X} \|Y - DX\|_F^2 \quad \text{s.t.} \quad \|\mathbf{x}_l\|_0 \le s,\ l = 1,\dots,N \\ \|\mathbf{d}_j\|_2 = 1,\ j = 1,\dots,n \tag{1}$$

where the number of atoms n and the sparsity level s are fixed. Here, $\|\cdot\|_2$ and $\|\cdot\|_0$ denote the ℓ_2 and ℓ_0 norm of a vector, respectively, and $\|\cdot\|_F$ is the Frobenius norm.

Problem (1) is NP-hard and admits multiple global optima; therefore the convergence to the global minimum is not guaranteed. In order to solve the DL problem, we follow the usual alternate optimization approach. More precisely, given the signal matrix Y and an initial dictionary D, at each iteration first the minimization problem in X is solved while D is fixed (**Sparse Coding** step) and then the minimization problem in D is solved while keeping X (possibly) fixed (**Dictionary Update** step).

The problem to be solved at the sparse coding step can be formulated as follows

$$\min_{X} \|Y-DX\|_F^2 \quad \text{s.t.} \quad \|\mathbf{x}_l\|_0 \le s,\ l = 1,\dots,N. \tag{2}$$

that can be decomposed in the solution of N problems, i.e. one for each signal

$$\min_{\mathbf{x}_l} \|\mathbf{y}_l - D\mathbf{x}_l\|_2^2 \quad \text{s.t.} \quad \|\mathbf{x}_l\|_0 \le s,\ l = 1,\dots,N. \tag{3}$$

For solving each problem (3), we employed Orthogonal Matching Pursuit (OMP), an iterative greedy algorithm that selects at each step the atom which is best correlated with the residual $\mathbf{e} := \mathbf{y} - D\mathbf{x}$. Then it produces a new approximation by projecting the signal $\mathbf{y}$ onto the dictionary elements that have already been selected ($D_\mathcal{S}$). We report in Algorithm 1 a naive version of OMP where the least squares solution $x_\mathcal{S}$ is computed from scratch at each step (refer to [4] for more details).

Algorithm 1 OMP (naive approach) [4]

```
Given $\mathbf{y} \in \mathbb{R}^m$, the sparsity level $s$, the dictionary $D \in \mathbb{R}^{m\times n}$ and the stopping tolerance $\epsilon > 0$
Initialize $\mathcal{S} = \emptyset$, $\mathbf{e} = \mathbf{y}$
while $|\mathcal{S}| < s$ and $\|\mathbf{e}\|_2 > \epsilon$ do
    $k = \text{argmax}_{j \notin \mathcal{S}} |\mathbf{e}^T \mathbf{d}_j|$
    $\mathcal{S} = \mathcal{S} \cup \{k\}$
    $\mathbf{x}_\mathcal{S} = (D_\mathcal{S}^T D_\mathcal{S})^{-1} D_\mathcal{S}^T \mathbf{y}$
    $\mathbf{e} = \mathbf{y} - D_\mathcal{S}\mathbf{x}_\mathcal{S}$
end while
```

Since at each step the current matrix $D_\mathcal{S}$ is updated by simply appending one column, a more efficient implementation can be obtained by exploiting the least squares solution just computed at the previous step. The most famous approaches make use of the Cholesky decomposition of $D_\mathcal{S}^T D_\mathcal{S}$ [4, sec. 2.2] or the QR decomposition of $D_\mathcal{S}$ [4, sec. 2.3]. Our computational experience showed that the OMP-QR implementation is faster when applied to DL [5]. Therefore, we implemented our parallel version of the OMP-QR code to speed-up the computational times.

Regarding the Dictionary Update step, the following minimization problem has to be solved

$$\min_{D,(X)} \|Y - DX\|_F^2 \quad \text{s.t.} \quad \|\mathbf{d}_j\|_2 = 1,\ j = 1,\dots,n \tag{4}$$

where the sparsity pattern of X is fixed. For this task we followed the K-SVD approach [6].

3. Dictionary Learning to reduce latency in Digital Twin

Reducing data latency is one of the main challenges within the DT context. This section aims to outline the proposed workflow, named DL4DT, to decrease data transmission time using DL as a compression technique. DL4DT, illustrated in Figure 1, takes place in two stages. First of all (Fig.1 top), the data are collected from the physical device, represented as a matrix Y and then transmitted to the digital counterpart. Here, the entire process of DL factorization is applied to Y, resulting in the learning of a reliable and robust overcomplete dictionary D and the sparse representation X. The dictionary D is both saved on the digital system and transmitted back to be saved also on the physical one. Afterwards, a new smaller dataset of signals Y_1 is collected (Fig.1 bottom). Instead of transferring the complete Y_1, we claim that computing its sparse representation X_1 with OMP using the reference dictionary D from stage 1 is sufficient. Transmitting X_1, which is highly sparse, indeed improves transmission time and reduces costs: solving a single Sparse Coding step demands fewer computational resources compared

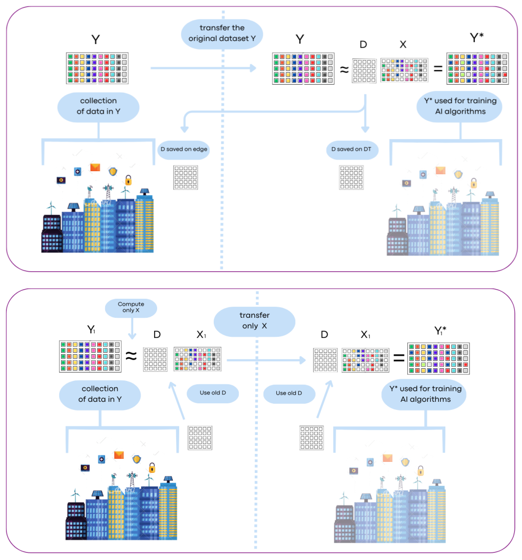

Figure 1: First (top) and next (bottom) runs of DL4DT.

to full DL, and transferring only X_1 is lighter than sending the entire Y_1. Indeed, suppose that Y_1 has N signals of m features each. Instead of passing all the $m \times N$ elements, with our method is enough to transmit the $s \times N$ non-zero elements of X_1. Notice that in sparse matrices, each non-zero element is stored as a triplet (row_index, column_index, non_zero_value) requiring a total storage of $s \times N \times 3$ values. Therefore, the benefit of transferring X_1 results in a reduction of $1 - \frac{3s}{m}$. Moreover, users have the flexibility to specify under which conditions the dictionary D has to be updated, in order to have more reliable results. For example, a reasonable choice can be updating the dictionary after a fixed period of time or when the accuracy of the AI algorithm on the compressed dataset starts to decrease too much. We refer to these conditions as *user_conditions* in the forthcoming Algorithm 2. As we will prove, DL4DT is very effective since DL techniques allow to massive compression preserving main important features of the dataset. DL4DT has been resumed in Algorithm 2.

4. Numerical Results

In this section, after introducing the datasets, we validate the DL approach as an effective compression tool for addressing DT latency problems. Then, we simulate and analyze the DL4DT workflow presented in Section 3, exploiting the DL ability to build a highly representative dictionary. All experiments were run on Galileo100 [7], an HPC infrastructure owned by CINECA with 528

Algorithm 2 DL4DT: workflow of a DT process with DL techniques.

```
Collect data on the physical counterpart in matrix $Y$.
Send $Y$ to the digital system.
Compute the dictionary $D$ and the sparse matrix $X$ with DL factorization of $Y$ on the digital system.
$i = 0$
while True do
    if $i = 0$ then
        Send the dictionary $D$ to the physical system and store it.
    else
        Compute $X$ using OMP-QR on the physical system.
        Send $X$ to the digital system.
    end if
    $i = i + 1$
    Compute $\tilde{Y} = DX$ on the digital system.
    Apply AI algorithm using $\tilde{Y}$ as dataset.
    if user_conditions then
        break
    end if
end while
```

computing nodes each 2 × CPU Intel CascadeLake 8260, with 24 cores each, 2.4 GHz, 384GB RAM and NVIDIA Mellanox Infiniband 100GbE network.

4.1. Datasets

We focused on three datasets with various types of data (images or timeseries) and dimensions: MNIST [8], FordA [9], and a fine-grained timeseries on the D.A.V.I.D.E. HPC system [10, 11]. D.A.V.I.D.E. is a supercomputer developed by E4 Computer Engineering [12] and hosted in the past by CINECA, with an integrated monitoring infrastructure called Examon [10]. In this work we focused on a subset of the data collected by Examon: for each of the 45 nodes, were considered 166 metrics such as core workloads, temperatures, fan speeds, power consumption, etc collected in 5-minute intervals. In detail, we focused on the 16th node.

4.2. Dictionary Learning compression

To evaluate the effectiveness of our compression, it is essential to compare the information generated by AI models trained on both the original and compressed datasets. This is crucial within the DT framework, where our primary aim is to extract valuable insights from compressed data.

We considered a CNN tailored for digit recognition [13] on MNIST dataset, a CNN able to perform anomaly detection suggested in [14] on FordA and an autoencoderbased model able to automatically detect anomalies in a semi-supervised fashion ([10, 11]) on D.A.V.I.D.E. After training the NNs described above on both original and compressed datasets, we compared their performance on the same test set by studying the *accuracy*, which is defined as the ratio of the number of correct predictions over the total number of predictions. Figure 2 compares respectively the test accuracy achieved by the NNs trained on the original dataset (green dotted line) and on a DL compression of MNIST (top) and FordA (bottom) concerning a sparsity level of $s = 50$ and a number of iterations $K = 20$ (orange solid line) across various compression levels. The results obtained with other settings of DL are shown in more detail in [5]. As expected,

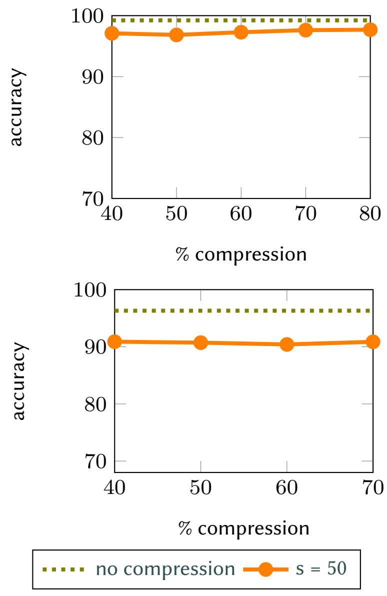

Figure 2: Accuracy of different compression levels with $s = 50$ compared to the accuracy with no compression on MNIST (top) and FordA dataset (bottom).

the accuracy computed on the compressed datasets is lower than the one computed on the original dataset. Despite not matching exactly the original accuracy, we still achieve extremely good results: with MNIST dataset we can even reach an accuracy of 97% with a compression of 80% against an accuracy of 99% with no compression, this is probably due to the redundant nature of the datasets, which makes it possible to achieve high accuracy levels even with high levels of compression. On FordA an overall accuracy of 91% is reached even with high compression levels against 96% with no compression. Figure 3 shows at the top the test accuracy achieved by the autoencoder trained on the original D.A.V.I.D.E dataset (green dotted line) and on the dataset compressed with DL with $s = 5$ and $K = 10$ (orange solid line) and different compression settings. The overall accuracy, approximately 86%, is lower than previous cases as expected due to the real-world nature of the dataset. However we notice that the test accuracy reached by training the autoencoder on the compressed training dataset is almost identical to the one obtained with no compression. However, when dealing with imbalanced datasets, it is better to consider the F-score value achieved for each class (normal signals and anomalies) rather than the accuracy. F-score value is defined as F-score:$= 2\,\frac{precision \times recall}{precision + recall}$, where $precision$ and $recall$ are the ratio of true positives to the total predicted positives and to the actual positives, respectively. We notice that the F-score reached on normal signals,

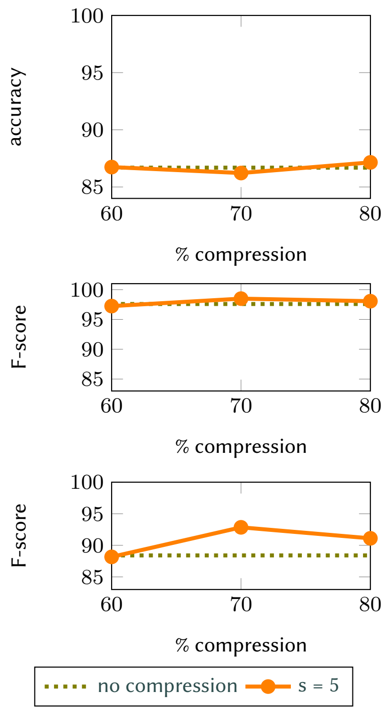

Figure 3: Accuracy (top), F-score on normal signals (middle) and on anomalies (bottom) with DL with $s = 5$ compared to the case with no compression on D.A.V.I.D.E. timeseries.

shown in the middle of Fig.3, remains almost unaffected by compression: across various DL configurations, the F-score consistently remains close to 98%, as the original case without compression. This finding aligns with our expectations, as the training set in this example consists only of signals without anomalies. As for the F-score of anomalies, shown at the bottom of Fig.3, we observe that this value increases when compression is more intense. Examining the details of the Recall and Precision values for these cases (Table 1), we notice that, respectively, the Recall for normal signals and the Precision for anomalies are higher compared to the case without compression.

Table 1
Precision and Recall values for normal signals and anomalies with no compression and 80% DL compression with $s = 5$.

compression	type of signal	Precision	Recall
0 %	normal	99.8	**95.4**
80 %	normal	99.8	**96.3**
0 %	anomaly	**79.8**	99.1
80 %	anomaly	**84.2**	99.1

These two values (Recall of normal signals and Precision of anomalies) take into account the cases where certain signals are identified as anomalies even though they are not. The higher the value, the more this type of error is avoided. Therefore, it is consistent that DL compression can increase these values, as DL is known as a valuable denoising tool, leading to improved anomaly detection.

Let us explore some implementations of the code. In our scenario, we have to deal with substantial problem dimensions but we can also benefit of the computational resources of an HPC cluster in the first stage of the workflow presented in Section 3. These resources can be fully employed in the OMP algorithm which can be parallelized with the Joblib python library [15] following what was mentioned in Section 2. Figure 4 illustrates the speedup achieved by executing OMP-QR both serially and in parallel with an increasing number of processors, where speedup is the ratio of the execution time of the serial code to the execution time of the parallel code performing the same task.

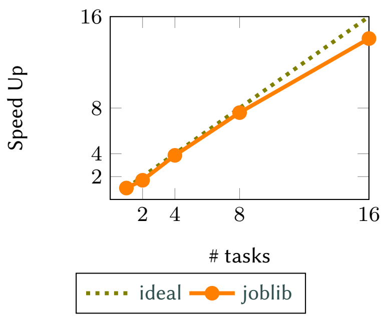

Figure 4: Speed up of OMP-QR algorithm in serial and with Joblib parallelization. For this type of problem it is not meaningful to increase resources beyond 16 tasks.

The proposed parallelization has a significant impact on the total computational time of the DL algorithm: when the plane DL algorithm is run sequentially with a single CPU, it requires about 20 hours to complete 20 iterations on a matrix of size 784×60.000, while the same algorithm implemented with the Joblib parallelized version of OMP-QR using 16 CPUs completes the task in about 5 hours. We have also developed a light C version of the OMP-QR code better suited for running on devices with limited computational resources.

4.3. Dictionary representativity

As already mentioned, the data provided by a DT do not usually show great variability. This section aims to verify whether the dictionary learned in the first stage is robust enough to accurately represent newly collected data. If successful, it would make it possible to run the sparse coding step (OMP-QR) without the need for a dictionary update. In particular we integrate the study of dictionary representativity into a simulation of the DL4DT workflow on D.A.V.I.D.E. dataset, keeping track of the original sizes, compression levels, and times.

The goal of the first stage is to learn a reliable and representative dictionary. Thus, we begin by considering the 4432 signals of its training set. In our workflow these data are sent to the digital twin where we choose to apply the strongest yet most meaningful compression, i.e. compression of 80 % with $s = 20$, $n = 349$ and 10 iterations. From previous studies we know that such a compression can reach an overall F-score level of about 97.9% on normal signals and 90.7% on anomalies, taking around 3 minutes. Then the dictionary is stored both in the digital twin and sent back to the physical one.
After a fixed time interval a new matrix of signals Y_1 is collected on the physical system. We simulate this new matrix of signals by taking the test set relative to the 16th node, since it is completely new to the dictionary and presents anomalies. We then compute its sparse representation matrix X_1 with a single run of OMP-QR with $s = 15$, taking around 3 seconds. The sparse representation matrix is then sent to the digital system where is used to reconstruct the signals as $\hat{Y}_1 = DX_1$. To evaluate the information loss due to the data compression we consider the autoencoder trained in the first run on the compressed train set and look if it is still able to detect the same anomalies testing it on the compressed test set $\hat{Y}_1$. We obtain extremely good results, achieving an F-score of 97% on normal samples and 89.9% on anomalies. These outcomes are very close to the results obtained without compression, which were respectively 97.9% and 90.7%. The DL setting that we choose is indeed a sensible choice: increasing the compression level contributes to smooth the signals with beneficial results, yet it remains highly representative with the sparsity level set to $s = 20$. We conduct a similar experiment using random compression, instead of DL, retaining only 30% of the samples chosen randomly from the test set, obtaining a F-score equal to 98% on normal samples and 63% on anomalies which is definitely worst. Thanks to this workflow, instead of transmitting the entire signal matrix Y_1 of dimensions 165×3074, is enough to compute and transfer its sparse representation X_1 which requires the

storage of $20 \times 3074 \times 3$ elements. This results in memory gain of 73%, requiring only 3 seconds and causing a minimal loss of information.

This process can be iterated multiple times, until the dictionary D requires updating to ensure more accurate outcomes. For instance, the dictionary might be refreshed periodically or whenever the performance of the AI algorithm on the compressed dataset begins to significantly decline. The results confirm that the dictionary D learned on the training set manages to represent new signals quite effectively. Indeed the accuracy levels achieved by the signals reconstructed with the old dictionary D are good, allowing a significant gain in computational efficiency.

5. Conclusions

The purpose of this work was to introduce a new efficient and lightweight compression tool within the Digital Twins framework that has minimal impact on the accuracy of AI models trained on compressed data (DL4DT). The numerical experiments showed that both with time-series and images the algorithm exhibited excellent behaviour, managing to compress the dataset up to 80% while preserving key information and therefore keeping the accuracy almost unchanged. As shown in Section 4.3, the dictionary learned from training data was able to represent new signals in an accurate manner in a sparse way. Moreover, in examples carried out on D.A.V.I.D.E. dataset turned out that such an algorithm also enhances data quality, serving as a potential preprocessing tool. Finally, due to the low computational cost of our parallel implementation of the OMP-QR, this approach allowed for on-device data compression, particularly useful with devices like IoT sensors, effectively reducing data exchange between devices while retaining the most crucial information. In conclusion, we can state that the DL compression algorithm effectively reduces the dataset memory demand, resulting in faster data transmission and reduced latency between distinct systems. Such a compression tool can have significant implications in Industry, where network infrastructures may not be high-performing but a wise and efficient use of digital twin systems is crucial for optimizing and managing production.

Acknowledgments

This work is supported by the EUROCC Italy National Competence Center. The Competence Center is part of EUROCC project funded by the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. A special thanks to the CINECA HPC department for their technical support. The work of MP is partially supported by INdAM-GNCS.

References

[1] R. Archibald, H. Tran, A dictionary learning algorithm for compression and reconstruction of streaming data in preset order, Discrete and Continuous Dynamical Systems - Series S 15 (2021). doi:10.3934/dcdss.2021102.

[2] A. Khelifati, M. Khayati, P. Cudré-Mauroux, Corad: Correlation-aware compression of massive time series using sparse dictionary coding, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 2289–2298. doi:10.1109/BigData47090.2019.9005580.

[3] B. Dumitrescu, P. Irofti, Dictionary Learning Algorithms and Applications, Springer Cham, 2018. doi:https://doi.org/10.1007/978-3-319-78674-2.

[4] B. Sturm, M. Christensen, Comparison of orthogonal matching pursuit implementations, EURASIP, 2012, pp. 220–224.

[5] L. Cavalli, Analysis and implementation of Dictionary Learning techniques in a Digital Twin framwork, Master thesis, University of Bologna, Bologna, Italy, 2023. Available at https://github.com/Eurocc-Italy/DL4DT.

[6] M. Aharon, M. Elad, A. Bruckstein, K-svd: An algorithm for designing overcomplete dictionaries for sparse representation, IEEE Transactions on Signal Processing 54 (2006) 4311–4322. doi:10.1109/TSP.2006.881199.

[7] Cineca, Galileo100, 2021. URL: https://www.hpc.cineca.it/systems/hardware/galileo100/.

[8] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Processing Magazine 29 (2012) 141–142. doi:10.1109/MSP.2012.2211477.

[9] J. Wichard, Classification of ford motor data (2009).

[10] A. Borghesi, A. Libri, L. Benini, A. Bartolini, Online anomaly detection in HPC systems, CoRR abs/1902.08447 (2019). URL: http://arxiv.org/abs/1902.08447. arXiv:1902.08447.

[11] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Anomaly detection using autoencoders in high performance computing systems, CoRR abs/1811.05269 (2018). URL: http://arxiv.org/abs/1811.05269. arXiv:1811.05269.

[12] E4 computer engineering., https://www.e4company.com/en/, 2024.

[13] F. Chollet, Simple mnist convnet, https://keras.io/examples/vision/mnist_convnet/, 2020.

[14] H. Fawaz, Timeseries classification from scratch, https://keras.io/examples/timeseries/timeseries_classification_from_scratch/, 2023.

[15] Joblib, https://github.com/joblib/joblib, 2023.