Data-Centric Engineering (2023), 4: e30
doi:10.1017/dce.2023.25

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

dCNN/dCAM: anomaly precursors discovery in multivariate time series with deep convolutional neural networks

Paul Boniol[1], Mohammed Meftah[2], Emmanuel Remy[2], Bruno Didier[3] and Themis Palpanas[1]

[1]LIPADE, Université Paris Cité, Paris, France
[2]PRISME, EDF R&D, Chatou, France
[3]DPN, EDF UNIE, Saint-Denis, France
Corresponding author: Paul Boniol; Email: boniol.paul@gmail.com

Received: 13 January 2023; **Revised:** 15 September 2023; **Accepted:** 19 October 2023

Keywords: anomaly detection; deep learning; explainability; precursors identification; time series

Abstract

Detection of defects and identification of symptoms in monitoring industrial systems is a widely studied problem with applications in a wide range of domains. Most of the monitored information extracted from systems corresponds to data series (or time series), where the evolution of values through one or multiple dimensions directly illustrates its health state. Thus, an automatic anomaly detection method in data series becomes crucial. In this article, we propose a novel method based on a convolutional neural network to detect precursors of anomalies in multivariate data series. Our contribution is twofold: We first describe a new convolutional architecture dedicated to multivariate data series classification; We then propose a novel method that returns dCAM, a dimension-wise Class Activation Map specifically designed for multivariate time series that can be used to identify precursors when used for classifying normal and abnormal data series. Experiments with several synthetic datasets demonstrate that dCAM is more accurate than previous classification approaches and a viable solution for discriminant feature discovery and classification explanation in multivariate time series. We then experimentally evaluate our approach on a real and challenging use case dedicated to identifying vibration precursors on pumps in nuclear power plants.

Impact Statement

Detection of defects and identification of symptoms in monitoring industrial systems is a widely studied problem with applications in a large range of domains. In this article, we propose a novel method, dCAM, based on a convolutional neural network, dCNN, to detect precursors of anomalies in multivariate data series. Experiments with several synthetic datasets demonstrate that dCAM is more accurate than previous classification approaches and a viable solution for discriminant feature discovery and classification explanation in multivariate time series. We then experimentally evaluate our approach on a real and challenging use case dedicated to identifying vibration precursors on pumps in nuclear power plants.

This research article was awarded an Open Materials badge for transparent practices. See the Data Availability Statement for details.

1. Introduction

Massive collections of data series (or time series) are becoming a reality in virtually every scientific and social domain, and there is an increasingly pressing need for relevant applications to develop techniques that can efficiently analyze them (Palpanas, 2015; Bagnall et al., 2019; Wang and Palpanas, 2021; Jakovljevic et al., 2022). Data series anomaly detection is a crucial problem with applications in a wide range of fields (Palpanas, 2015; Boniol et al., 2022; Paparrizos et al., 2022a, 2022b), that all share the same well-studied goal (Barnet and Lewis, 1994; Subramaniam et al., 2006; Yeh et al., 2016): detecting anomalies as fast as possible to avoid any critical event. Such applications can be found in biology, astronomy, and engineering areas. Some of these sectors are well-studied and theoretically well-explored. The knowledge acquired by the expert can be used to build an algorithm that efficiently detects any kind of behavior derived from a potential well-defined *normality.* However, such algorithms can be difficult to concretize and might have difficulty adapting to unknown or unclear changes over time. On the other side, if the data available are representative enough to correctly illustrate the system's health state, a data-driven method could provide more flexibility. For instance, in the case of fraud detection, a knowledge-based model looks for known frauds, while a data-driven model might be helpful in finding new patterns, which is crucial since frauds can change frequently and dynamically. The same statement can be made in the general case of anomaly or outlier detection.

1.1. Anomaly detection primer

First, one should note that no single, universal definition of outliers or anomalies exists. In general, an anomaly is an observation that appears to deviate markedly from other members of the sample in which it occurs. This fact may raise suspicions that the specific observation was generated by a different mechanism than the remainder of the data. This mechanism may be an erroneous data measurement and collection procedure or an inherent variability in the domain of the data under inspection. Nevertheless, such observations are interesting in both cases, and the analyst would like to know about them. The latter can be tackled by either looking at single values or a sequence of points. In the specific context of points, we are interested in finding points that are far from the normal distribution of values that correspond to *healthy* states. In the specific context of sequences, we are interested in identifying anomalous subsequences that are, unlike an outlier, not a single abnormal value but rather an abnormal evolution of values. This work will study the specific case of subsequence anomaly detection in data series.

As usually addressed in the literature (Breunig et al., 2000; Keogh et al., 2007; Liu et al., 2008; Liu et al., 2009; Senin et al., 2015; Yeh et al., 2016), one can decide to adopt a fully unsupervised method. These approaches benefit from working without needing any collection of known and tagged anomalies and can automatically detect unknown abnormal behaviors. Several approaches have been proposed. For example, general methods for multi-dimensional points outlier detection have been proposed (Breunig et al., 2000; Liu et al., 2008; Ma et al., 2020). Nevertheless, algorithms such as Isolation forest (Liu et al., 2008) seem to perform well for the specific case of subsequence anomaly detection (Boniol and Palpanas, 2020). Moreover, recently proposed approaches, such as NormA (Boniol et al., 2021) (that aims to build a weighted set of centroids summarizing the different recurrent subsequences in the data series) and Series2Graph (Boniol and Palpanas, 2020) (that aims to build a directed graph in which a trajectory corresponds to a subsequence in the time series), model the normal behaviors of the data series and have shown to outperform the previous state-of-the-art approaches.

1.2. Supervised detection of anomaly precursors

In the previous section, anomalies were considered unknown. However, one can assume that experts know precisely which event they want to detect and have a data series collection corresponding to these anomalies. In that case, we have a database of anomalies at our disposal. As a consequence, one can decide to adopt supervised methods. A question that naturally arises is the following: is it possible to detect subsequences that happened before the known anomaly that might lead to an explanation of the anomaly

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

(and potentially predict it)? Such subsequences can be called *precursors* or *symptoms* of anomalies. Usually used in medical domains, we can infer the medical definition and adapt it to the task of anomaly detection in large data series. We will use the generic term *precursors* in the remainder of the article. Detecting such subsequences might be significantly helpful for knowledge experts to prevent future anomalies from occurring or understand why an anomaly occurred (or facilitate its understanding). In several engineering applications, it is required to analyze measurements from many sensors (across many different locations) to detect potential failures. Being able to detect failures is not enough, and identifying which sensors (and which timestamps) are related to the failure provides essential information on the origin of the failure. For instance, in the monitoring of nuclear power plants, it is as important to detect the vibration of a given pump as to identify precursors and unusual measurements of sensors within the plant that could explain or prevent future vibrations. Thus, the task is to detect (in a supervised manner) known anomalies and retrieve potential precursors. We now propose a formal definition of the problem mentioned above.

Problem 1 (Precursors Identification Problem Definition). *Given a monitored system* $\mathcal{M}$, *a set of data series* $T^{\mathcal{N}}_{\mathcal{M}}$ *that represents the healthy state of* $\mathcal{M}$ (*healthy state labeled* $\mathcal{N}$), *a set of data series* $T^{\mathcal{A}}_{\mathcal{M}}$ *that represents the state of* $\mathcal{M}$ *before an anomalous state (anomalous state labeled* $\mathcal{A}$*), we first have to find a function* f *that takes as input* $T^{\mathcal{A}}_{\mathcal{M}}$ *and* $T^{\mathcal{N}}_{\mathcal{M}}$, *and returns* $s \in \{\mathcal{N}, \mathcal{A}\}$. *We then have to find a function* g *that takes as input* $T^{\mathcal{A}}_{\mathcal{M}}$ *and* f, *and returns* $S \subset T^{\mathcal{A}}_{\mathcal{M}}$ (S *being the set of subsequences in* $T^{\mathcal{A}}_{\mathcal{M}}$ *precursors of the upcoming anomalies). Formally,* f *and* g *can be written as follows:* $f : T^{\mathcal{N}}_{\mathcal{M}}, T^{\mathcal{A}}_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$ *and* $g : T^{\mathcal{A}}_{\mathcal{M}}, f \rightarrow S$.

As Problem 1 is defined as a supervised task, one can decide to use time series classification approaches. The latter can be performed using either (1) pre-extracted features or (2) raw values of the time series.

For the first category (1), the main idea is to use the dataset of time series (or subsequences of time series) to create a dataset whose samples are described by features common to all-time series. For the specific case of time series, the feature extraction step can be performed using TSFresh Python library (Christ et al., 2018) (Time Series Feature extraction based on scalable hypothesis tests). The latter is used for automated time series feature extraction and selection based on the FRESH algorithm (Christ et al., 2016). More specifically, it automatically selects relevant features for a specific task. This is achieved using statistical tests, time series heuristics, and machine learning algorithms. Then, using the feature-based dataset, standard machine learning classifiers can be employed to classify each time series. Examples of traditional machine learning classifiers are Support Vector Classifier (SVC) (Boser et al., 1992) (i.e., classifiers that map instances in space in order to maximize the width of the gap between the classes), naive Bayes classifier (Zhang, 2004) (i.e., classifiers based on Bayes' theorem to predict the class of a new instance based on prior probabilities and class-conditional probabilities), Multi-Layer Perceptron (MLP) (Hinton, 1989) (i.e., fully connected neural networks), AdaBoost (Freund and Schapire, 1995) (i.e., boosting ensemble machine learning algorithms), or Random Forest Classifier (Ho, 1995) (ensemble machine learning algorithm that combines multiple decision trees, where each tree is built using a random subset of the features and a random sample of the data). On top of the aforementioned classifiers, explainability frameworks, such as LIME (Ribeiro et al., 2016) and SHAP (Lundberg and Lee, 2017) can be used in order to identify which features have the most important effect on the classification prediction. However, feature-based classification and explanation methods can be limited when applied to time series. Whereas features are efficient for summarizing time series datasets (e.g., setting a constant number of features for variable length time series), it might miss important information in the shape of consecutive values, which may be crucial for anomaly detection and precursor identification. Moreover, the choice of features heavily impacts the classification accuracy and is not desirable in tasks related to precursors identification with unknown properties. Therefore, using methods that do not need any feature selection step is preferable when working on agnostic scenarios.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

For methods based on raw values of the time series (ii), standard data series classification methods are based on distances to the instances' nearest neighbors, with k-NN classification (using the Euclidean or Dynamic Time Warping (DTW) distances) being a popular baseline method (Dau et al., 2019). Nevertheless, recent works have shown that ensemble methods using more advanced classifiers achieve better performance (Bagnall et al., 2015; Lines et al., 2016). Following recent breakthroughs in the computer vision community, new studies successfully propose deep learning methods for data series classification (Chen et al., 2013; Zheng et al., 2014; Cui et al., 2016; Le Guennec et al., 2016; Zhao et al., 2017; Wang et al., 2018; Fawaz et al., 2019), such as Convolutional Neural Network (CNN) and Residual Neural Network (ResNet) (Wang et al., 2017). For some CNN-based models, the Class Activation Map (CAM) (Zhou et al., 2016) can be used as an explanation for the classification result. CAM has been used for highlighting the parts of an image that contribute the most to a given class prediction and has also been adapted to data series (Wang et al., 2017; Fawaz et al., 2019). Thus, CAM can be used to identify subsequences that contribute the most to the anomaly prediction and, consequently, to solve Problem 1. This article considers Convolutional Neural Network joined with the Class Activation Map as a strong baseline.

1.3. Limitations of previous approaches

As regards existing methods that can solve Problem 1, CNN/CAM for data series suffers from one important limitation. CAM is a weighted mapping technique that returns a univariate series (of the same length as the input instances) with high values aligned with the subsequences of the input that contribute the most to a given class identification. Thus, in the case of multivariate data series as input, no information can be retrieved from CAM on which dimension is contributing. Therefore, precursors within specific dimensions cannot be retrieved using existing methods.

1.4. Contributions

In this article, we will focus on solving Problem 1. First, we explore the possibility of using a supervised time series classification algorithm to solve precursor discovery tasks. We then propose a novel method, dCNN/dCAM, that overcomes the limitations of previous supervised algorithms. Moreover, dCNN/dCAM permits the identification of specific patterns (without any prior knowledge) and alerts the expert on the imminent occurrence of the anomaly. Finally, we demonstrate this latter claim in a real industrial use case. The article is structured as follows:

- *Section 2*: We first describe the notations and the concepts related to time series and neural networks. We then describe the usual Class Activation Map and explain its limitations regarding multivariate data series in detail.
- *Section 3*: We describe a new convolutional architecture, dCNN, that enables the comparison of dimensions by changing the structure of the network input and using two-dimensional convolutional layers. We then introduce dCAM, a novel method that takes advantage of dCNN and returns a multivariate CAM that identifies the important parts of the input series for *each* dimension.
- *Section 4:* We then experimentally evaluate the classification accuracy of dCNN/dCAM over CNN/CAM on a synthetic benchmark and demonstrate the benefit of our proposed approach.
- *Section 5:* We experimentally evaluate our proposed approach over a real industrial use case.

2. Background and related work

[Data Series] A multivariate, or D-dimensional data series $T \in \mathbb{R}^{(D,n)}$ is a set of D univariate data series of length n. We note $T = \left[T^{(0)}, \ldots, T^{(D-1)}\right]$ and for $j \in [0, D-1]$, we note the univariate data series $T^{(j)} = \left[T_0^{(j)}, T_1^{(j)}, \ldots, T_{n-1}^{(j)}\right]$. A subsequence $T_{i,\ell}^{(j)} \in \mathbb{R}^{\ell}$ of the dimension $T^{(j)}$ of the multivariate data series

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

T is a subset of contiguous values from $T^{(j)}$ of length ℓ (usually $\ell \ll n$) starting at position i; formally, $T^{(j)}_{i,\ell} = \left[T^{(j)}_i, T^{(j)}_{i+1}, \ldots, T^{(j)}_{i+\ell-1}\right]$.

[Neural Network Notations] We are interested in classifying data series using a neural network architecture model. We define the neural network input as $X \in \mathbb{R}^n$ for univariate data series (with x_i the i^{th} value and $X_{i,\ell}$ the sequence of ℓ values following the i^{th} value), and $\mathbf{X} \in \mathbb{R}^{(D,n)}$ for multivariate data series (with $x_{j,i}$ the i^{th} value on the j^{th} dimension and $\mathbf{X}_{j,i,\ell}$ the sequence of ℓ values following the i^{th} value on the j^{th} dimension).

Dense Layer: The basic layer of neural networks is a fully connected layer (also called dense layer) in which every input neuron is weighted and summed before passing through an activation function. For univariate data series, given an input data series $X \in \mathbb{R}^n$, given a vector of weights $W \in \mathbb{R}^n$ and a vector $B \in \mathbb{R}^n$, we have:

$$h = f_a\left(\sum_{x_i, w_i, b_i \in (X, W, B)} w_i * x_i + b_i\right). \tag{1}$$

f_a is called the activation function and is a nonlinear function. The commonly used activation function f_a is the rectified linear unit (ReLU) (Nair and Hinton, 2010) that prevents the saturation of the gradient (other functions that have been proposed are Tanh and Leaky ReLU (Xu et al., 2015)). For the specific case of multivariate data series, all dimensions are concatenated to give input $X, W \in \mathbb{R}^{D*n}$. Finally, one can decide to have several output neurons. In this case, each neuron is associated with a different W and B, and equation (1) is executed independently.

Convolutional Layer: This layer has played a significant role in image classification (Krizhevsky et al., 2012; LeCun et al., 2015; Wang et al., 2017), and recently for data series classification (Fawaz et al., 2019). Formally, for multivariate data series, given an input vector $\mathbf{X} \in \mathbb{R}^{(D,n)}$, and given matrices weights $\mathbf{W}, \mathbf{B} \in \mathbb{R}^{(D,\ell)}$, the output $h \in \mathbb{R}^n$ of a convolutional layer can be seen as a univariate data series. The tuple (W, B) is also called a kernel, with (D, ℓ) the size of the kernel. Formally, for $h = [h_0, \ldots, h_n]$, we have:

$$h_i = f_a\left(\sum_{X^{(j)}, W^{(j)}, B^{(j)} \in (\mathbf{X}, \mathbf{W}, \mathbf{B})} \sum_{x_k, w_k, b_k \in \left(X^{(j)}_{i-\lfloor \frac{\ell}{2} \rfloor, i+\lfloor \frac{\ell}{2} \rfloor}, W^{(j)}, B^{(j)}\right)} w_k * x_k + b_k\right). \tag{2}$$

In practice, we have several kernels of size (D, ℓ). The result is a multivariate series with dimensions equal to the number of kernels, n_f. For a given input $\mathbf{X} \in \mathbb{R}^{(D,n)}$, we define $A \in \mathbb{R}^{(n_f, n)}$ to be the output of a convolutional layer $\mathrm{conv}(n_f, \ell)$. A_m is thus a univariate series corresponding to the output of the m^{th} kernel. We denote with $A_m(T)$ the univariate series corresponding to the output of the m^{th} kernel when T is used as input.

Global Average Pooling Layer: Another type of layer frequently used is pooling. Pooling layers compute average/max/min operations, aggregating values of previous layers into a smaller number of values for the next layer. A specific type of pooling layer is Global Average Pooling (GAP). This operation averages an entire output of a convolutional layer $A_m(T)$ into one value, thus providing invariance to the position of the discriminative features.

Learning Phase: The learning phase uses a loss function $\mathcal{L}$ that measures the accuracy of the model and optimizes the various weights. For the sake of simplicity, we note Ω the set containing all weights (e.g., matrices $\mathbf{W}$ and $\mathbf{B}$ defined in the previous sections). Given a set of instances $\mathcal{X}$, we define the average loss as: $J(\Omega) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{X} \in \mathcal{X}} \mathcal{L}(\mathbf{X})$. Then for a given learning rate α, the average loss is back-propagated to all weights in the different layers. Formally, back-propagation is defined as follows:

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

$\forall \omega \in \Omega, \omega \leftarrow \omega - \alpha \frac{\partial J}{\partial \omega}$. In this article, we use the stochastic gradient descent using the ADAM optimizer (Kingma and Ba, 2015) and cross-entropy loss function.

2.1. Convolutional-based neural network

We now describe the standard architectures used in the literature. The first is Convolutional Neural Networks (CNNs) (Wang et al., 2017; Fawaz et al., 2019). CNN is a concatenation of convolutional layers (joined with ReLU activation functions and batch normalization). The last convolutional layer is connected to a Global Average Pooling layer and a dense layer. In theory, instances of multiple lengths can be used with the same network. A second architecture is the Residual Neural Network (ResNet) (Wang et al., 2017; Fawaz et al., 2019). This architecture is based on the classical CNN, to which we add residual connections between successive blocks of convolutional layers to prevent the gradients from exploding or vanishing. Other methods have been proposed in the literature (Cui et al., 2016; Serrà et al., 2018; Fawaz et al., 2019; Ismail Fawaz et al., 2020), though CNN and ResNet have been shown to perform the best for multivariate time series classification (Fawaz et al., 2019). InceptionTime (Ismail Fawaz et al., 2020) has not been evaluated on multivariate data series but demonstrated state-of-the-art performance on univariate data series.

2.2. Class activation map

Once the model is trained, we need to find the discriminative features that led the model to decide which class to attribute to each instance. Class Activation Map (Zhou et al., 2016) (CAM) has been proposed to highlight the parts of an image that contributes the most to a given class identification. The latter has been experimented on data series (Wang et al., 2017; Fawaz et al., 2019) (univariate and multivariate). This method explains the classification of a certain deep learning model by emphasizing the subsequences that contribute the most to a certain classification. Note that the CAM method can only be used if (1) a Global Average Pooling layer has been used before the softmax classifier, and (2) the model accuracy is high enough. Thus, only the standard architectures CNN and ResNet proposed in the literature can benefit from CAM. We now define the CAM method (Wang et al., 2017; Fawaz et al., 2019). For an input data series T, let $A(T)$ be the result of the last convolutional layer $\text{conv}(n_f, \ell)$, which is a multivariate data series with n_f dimensions and of length n. $A_m(T)$ is the univariate time series for the dimension $m \in [1, n_f]$ corresponding to the m^{th} kernel. Let $w_m^{\mathcal{C}_j}$ be the weight between the m^{th} kernel and the output neuron of class $\mathcal{C}_j \in \mathcal{C}$. Since a Global Average Pooling layer is used, the input to the neuron of class $\mathcal{C}_j$ can be expressed by the following equation:

$$z_{\mathcal{C}_j}(T) = \sum_m w_m^{\mathcal{C}_j} \sum_{A_{m,i}(T) \in A_m(T)} A_{m,i}(T). \tag{3}$$

The second sum represents the averaged time series over the whole time dimension. Note that weight $w_m^{\mathcal{C}_j}$ is independent of index i. Thus, $z_{\mathcal{C}_j}$ can also be written by the following equation:

$$z_{\mathcal{C}_j}(T) = \sum_{A_{m,i}(T) \in A_m(T)} \sum_m w_m^{\mathcal{C}_j} A_{m,i}(T). \tag{4}$$

Finally, $\text{CAM}_{\mathcal{C}_j}(T) = [\text{CAM}_{\mathcal{C}_j,0}(T), \ldots, \text{CAM}_{\mathcal{C}_j,n-1}(T)]$ that underlines the discriminative features of class $\mathcal{C}_j$ is defined as follows:

$$\forall i \in [0, n-1], \text{CAM}_{\mathcal{C}_j,i}(T) = \sum_m w_m^{\mathcal{C}_j} A_{m,i}(T). \tag{5}$$

As a consequence, $\text{CAM}_{\mathcal{C}_j}(T)$ is a weighted mapping technique that returns a univariate data series where each element at index i indicates the significance of the index i (regardless of the dimensions) for the classification as class $\mathcal{C}_j$. Figure 1a depicts the process of computing CAM and finding the discriminant subsequences in the initial series.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

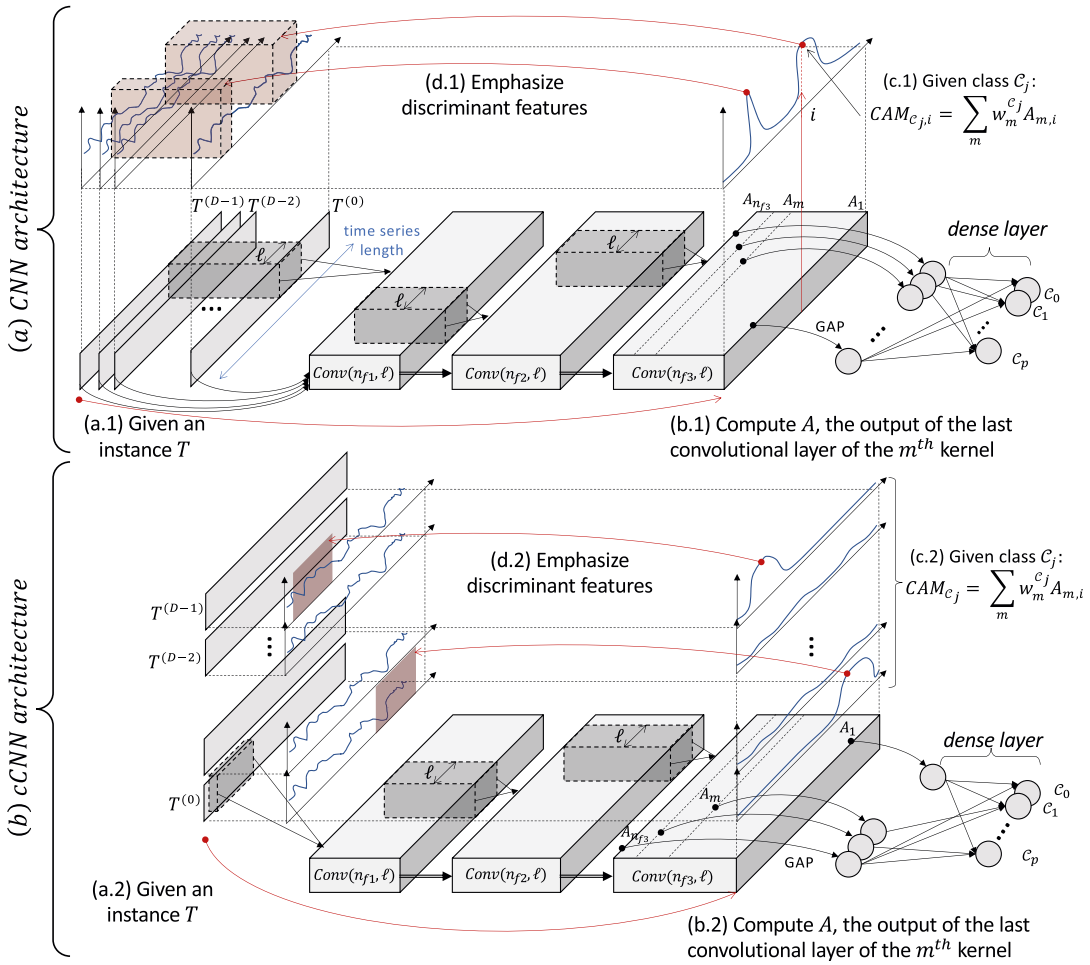

Figure 1. *Illustration of Class Activation Map for (a) CNN architecture and (b) cCNN architecture with three convolutional layers (n_{f1}, n_{f2}, and n_{f3} different kernels respectively of size all equal to ℓ).*

2.3. CAM limitations for multivariate series

As mentioned earlier, a CAM that highlights the discriminative subsequences of class $\mathcal{C}_j$, $\text{CAM}_{\mathcal{C}_j}(T)$, is a weighted mapping technique that returns a univariate data series. The information provided by $\text{CAM}_{\mathcal{C}_j}(T)$ is sufficient for the case of univariate series classification, but not for multivariate series classification. Even though the significant temporal index may be correctly highlighted, no information can be retrieved on which dimension is significant or not. Solving this serious limitation is a significant challenge in several domains. For that purpose, one can propose rearranging the input structure of the network so that the CAM becomes a multivariate data series. A new solution would be to decide to use a 2D convolutional neural network with kernel size $(\ell, 1)$, such that each kernel slides on each dimension separately. Thus, for an input data series T, $\mathbf{A}_m(T)$ would become a multivariate data series for the variable $m \in [1, n_f]$, and $A_m^{(d)}(T) \in \mathbf{A}_m(T)$ would be a univariate time series that would correspond to the dimension d of the initial data series. We call this solution *cCNN*, and we use *cCAM* to refer to the corresponding Class Activation Map. Figure 1b illustrates cCNN architecture and cCAM. Note that if a GAP layer is used, then architectures other than CNN can be used, such as ResNet and InceptionTime. We denote these baselines as *cResNet* and *cInceptionTime*.

Nevertheless, new limitations arise from this solution. First, the dimensions are not compared together: Each kernel of the input layer will take as input only one of the dimensions at a time. Thus, features depending on more than one dimension will not be detected.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

Recent works study the specific case of multivariate data series classification explanation. A benchmark study analyzing the saliency/explanation methods for multivariate time series concluded that the explainable methods work better when the multivariate data series is handled as an image (Ismail et al., 2020), such as in the CNN architecture. This confirms the need to propose a method specifically designed for multivariate data series. Finally, some recently proposed approaches (Assaf et al., 2019; Hsieh et al., 2021) address the problems of identifying the discriminant features and discriminant temporal windows independently from one another. For instance, MTEX-CNN (Assaf et al., 2019) is an architecture composed of two blocks. The first block is similar to cCNN. The second block consists of merging the results of the first block into a 1D convolutional layer, which enables comparing dimensions. A variant of CAM (Selvaraju et al., 2017) is applied to the last convolutional layer of the 1st block in order to find discriminant features for each dimension. The discriminant temporal windows are detected with the CAM applied to the last convolutional layer of the second block. In practice, however, this architecture does not manage to overcome the limitations of cCNN: discriminant features that depend on several dimensions are not correctly identified by MTEX-CNN, which has similar accuracy to cCNN (we elaborate on this in Section 4.2).

In our experimental evaluation, we compare our approach to the MTEX-CNN, cCNN, cResNet, and cInceptionTime, and further demonstrate their limitations when addressing the problem at hand.

3. Proposed approach

In this section, we describe our proposed approach, dCAM (dimension-wise Class Activation Map). Based on a new architecture that we call dCNN, (as well as variant architectures such as dResNet and dInceptionTime), dCAM aims to provide a multivariate CAM pointing to the discriminant features within each dimension. Contrary to the previously described baseline (cCNN, cResNet, and cInceptionTime), one kernel on the first convolutional layer will take as input all the dimensions together with different permutations. Thus, similarly to the standard CNN architecture, features depending on more than one dimension will be detectable while still having a multivariate CAM. Nevertheless, the latter has to be processed such that the significant subsequences are detected.

We first describe the proposed architecture dCNN that we need in order to provide a dimension-wise Class Activation Map (dCAM) while still being able to extract multivariate features. We then demonstrate that the transformation needed to change CNN to dCNN can also be applied to other, more sophisticated architectures, such as ResNet and InceptionTime, which we denote as dResNet and dInceptionTime. We demonstrate that using permutations of the input dimensions makes the classification more robust when important features are localized into small subsequences within some specific dimensions.

We then present in detail how we compute dCAM (based on a dCNN). Our solution benefits from the permutations injected into the dCNN to identify the most discriminant subsequences used for the classification decision.

3.1. Dimension-wise architecture

As mentioned earlier, the classical CNN architecture mixes all dimensions in the first convolutional layer. Thus, the CAM is a univariate data series and does not provide any information on which dimension is the discriminant one for the classification. To address this issue, we can use a two-dimensional CNN architecture by re-organizing the input (i.e., the cCNN solution we described earlier). In this architecture, one kernel (of size $(1,\ell,1)$) slides on each dimension independently. Thus, for a given data series $\left(T^{(0)},\ldots,T^{(D-1)}\right)$ of length n, the convolutional layers return an array of three dimensions $\left(n_f,D,n\right)$, each row $m\in[0,D]$ corresponding to the extracted features on dimension m. Nevertheless, the kernels $(1,\ell,1)$ get as input each dimension independently. Evidently, such an architecture cannot learn features that depend on multiple dimensions.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

3.2. A first architecture: dCNN

In order to have the best of both cases, we propose the dCNN architecture, where we transform the input into a cube, in which each row contains a given combination of all dimensions. One kernel (of size $(D, \ell, 1)$) slides on all dimensions D times. This allows the architecture to learn features on multiple dimensions simultaneously. Moreover, the resulting CAM is a multivariate data series. In this case, one row of the CAM corresponds to a given combination of the dimensions. However, we still need to be able to retrieve information for each dimension separately, as well. To do that, we ensure that each row contains a different permutation of the dimensions. As the weights of the kernels are at fixed positions (for specific dimensions), a permutation of the dimensions will result in a different CAM. Formally, for a given data series T, we note $C(T) \in \mathbb{R}^{(D,D,n)}$ the input data structure of dCNN, defined as follows:

$$C(T) = \begin{pmatrix} T^{(D-1)} & T^{(0)} & \dots & T^{(D-3)} & T^{(D-2)} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ T^{(1)} & T^{(2)} & \dots & T^{(D-1)} & T^{(0)} \\ T^{(0)} & T^{(1)} & \dots & T^{(D-2)} & T^{(D-1)} \end{pmatrix}. \tag{6}$$

Note that each row and column of $C(T)$ contains all dimensions. Thus, a given dimension $T^{(i)}$ is never at the same position in $C(T)$ rows. The latter is a crucial property for the computation of dCAM. In practice, we guarantee the latter property by shifting the order of the dimensions by one position. For instance, in equation (6), the dimension order of the first row is $\left[T^{(0)}, T^{(1)}, \dots, T^{(D-2)}, T^{(D-1)}\right]$ (i.e., the first dimension of T is at the first position in the row and the last dimension of T is at the last position in the row), and the dimension order of the second row is $\left[T^{(1)}, T^{(2)}, \dots, T^{(D-1)}, T^{(0)}\right]$ (i.e., the first dimension of T is now at the last position in the row, and the second dimension of T is now at the first position in the row). Thus $T^{(0)}$ in the first row is aligned with $T^{(1)}$ in the second row. A different order of T dimensions will thus generate a different matrix $C(T)$.

Figure 2 depicts the dCNN architecture. The input $C(T)$ is forwarded into a classical two-dimensional CNN. The rest of the architecture is independent of the input data structure. Similarly, the training procedure can be freely chosen by the user. For the rest of the article, we will use the cross-entropy loss function and the ADAM optimizer.

Observe that multiple permutations of the original multivariate series (provided only by the different rows of $C(T)$) will be processed by several convolutional filters, enabling the kernel to examine multiple different combinations of dimensions and subsequences. Note that the kernels of the dCNN will be sparse, which has a significant impact on overfitting.

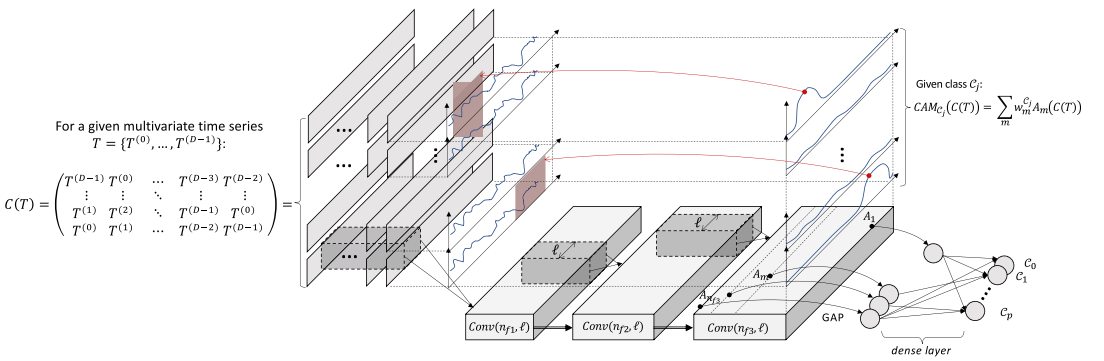

Figure 2. *dCNN architecture and application of the CAM.*

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

3.3. Variant architectures: dResNet and dInceptionTime

As mentioned earlier, any architecture using a GAP layer after the last convolutional layer can benefit from dCAM. Thus, different (and more sophisticated) architectures can be used with our approach. To that effect, we propose two new architectures dResNet and dInceptionTime, based on the state-of-the-art architectures ResNet (Wang et al., 2017) and InceptionTime (Ismail Fawaz et al., 2020). The transformations that lead to dResNet and dInceptionTime are very similar to that from CNN to dCNN, using $C(T)$ as input to the transformed networks. The convolutional layers are transformed from 1D (as proposed in the original architecture (Wang et al., 2017; Ismail Fawaz et al., 2020)) to 2D. Similarly to dCNN, the kernel sizes are $(D, \ell, 1)$ and convolute over each row of $C(T)$ independently.

We demonstrate in the experimental section that these architectures do not affect the usage of our proposed approach dCAM, and we evaluate the choice of architecture on both classification and discriminant feature identification. In the following sections, we describe our methods assuming the dCNN architecture. Nevertheless, it works exactly the same for the other two architectures.

3.4. Dimension-wise class activation map

At this point, we have our network trained to classify instances among classes $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_p$. We now describe in detail how to compute dCAM that will identify discriminant features within dimensions. We assume that the network has to be accurate enough in order to provide a meaningful dCAM. At first glance, one can compute the regular Class Activation Map $\mathrm{CAM}_{\mathcal{C}_j}(C(T)) = \sum_m w_m^{\mathcal{C}_j} A_m(C(T))$. However, a high value on the i^{th} row at position t on $\mathrm{CAM}_{\mathcal{C}_j}(C(T))$ does not mean that the subsequence at position t on the i^{th} dimension is important for the classification. It instead means that the combination of dimensions at the i^{th} row of $C(T)$ is important.

3.4.1. Random permutation computations

Given those different combinations of dimensions (i.e., one row of $C(T)$) produce different outputs (i.e., the same row in $\mathrm{CAM}_{\mathcal{C}_j}(C(T))$), the positions of the dimensions within the $C(T)$ rows have an impact on the Class Activation Map. Consequently, for a given combination of dimensions, we can assume that at least one dimension at a given position is responsible for the high value in the Class Activation Map row. For the remainder of this article, we use Σ_T as the set of all possible permutations of T dimensions, and $S_T^i \in \Sigma_T$ for a single permutation of T. For example, for a given data series $T = \{T^{(0)}, T^{(1)}, T^{(2)}\}$, one possible permutation is $S_T^i = \{T^{(1)}, T^{(0)}, T^{(2)}\}$.

Figure 3 depicts an example of Class Activation Maps for different permutations. In this figure, for three given permutations of T (i.e., S_T^0, S_T^1, and S_T^2), we notice that when $T^{(2)}$ is in position two of the second row of $C(S_T^i)$, the Class Activation Map $\mathrm{CAM}(C(S_T^i))$ is greater than when $T^{(2)}$ is not in position two. We infer that the second dimension of T in position two is responsible for the high value. Thus, we may examine different dimension combinations by keeping track of which dimension at which position is activating the Class Activation Map the most. In the remainder of this section, we describe the steps necessary to retrieve this information.

Definition 1. For a given data series $T = \{T^{(0)}, T^{(1)}, \ldots, T^{(D-1)}\}$ of length n and its input data structure $C(T)$, we define function idx, such that $\mathrm{idx}(T^{(i)}, p_j)$ returns the row indices in $C(T)$ that contain the dimension $T^{(i)}$ at position p_j.

We can now define the following transformation $\mathcal{M}$.

Definition 2. For a given data series $T = \{T^{(0)}, T^{(1)}, \ldots, T^{(D-1)}\}$ of length n, a given class $\mathcal{C}_j$ and Class Activation Map, we define $\mathcal{M}(\mathrm{CAM}_{\mathcal{C}_j}(C(T))) \in \mathbb{R}^{(D,D,n)}$ (with $\mathrm{CAM}_{\mathcal{C}_j}(C(T)) \in \mathbb{R}^{(D,n)}$ and $\mathrm{CAM}_{\mathcal{C}_j}(C(T))_i$ its i^{th} row) as follows:

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

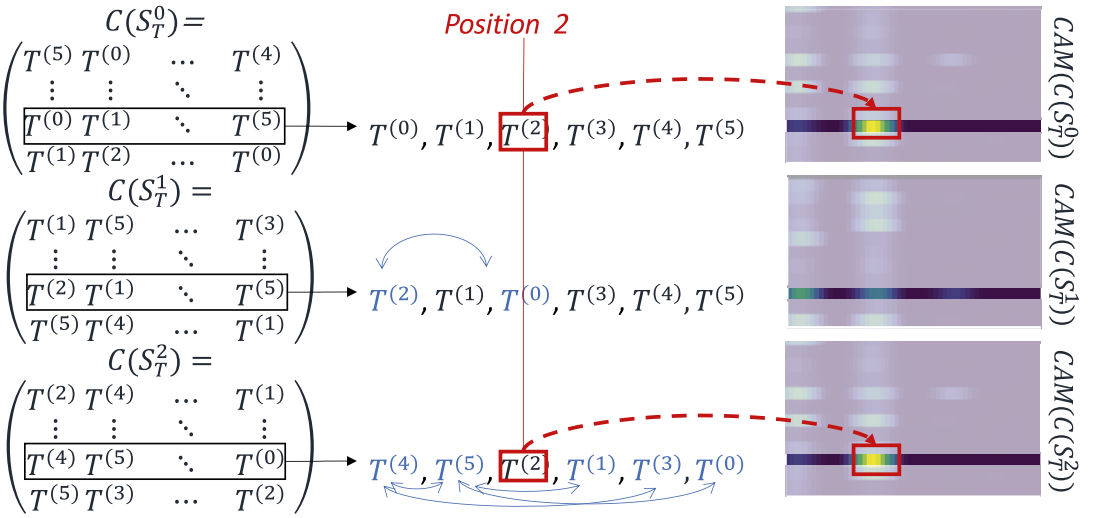

Figure 3. *Example of Class Activation Map results for different permutations.*

$$\mathcal{M}\left(\mathrm{CAM}_{\mathcal{C}_j}(C(T))\right) = \begin{pmatrix} \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(0)},0\right)} & \cdots & \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(0)},D-1\right)} \\ \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(1)},0\right)} & \cdots & \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(1)},D-1\right)} \\ \vdots & \cdots & \vdots \\ \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(D-1)},0\right)} & \cdots & \mathrm{CAM}_{\mathcal{C}_j}(C(T))_{\mathrm{idx}\left(T^{(D-1)},D-1\right)} \end{pmatrix}. \tag{7}$$

Figure 4 depicts the $\mathcal{M}$ transformation. As explained in Definition 2, the $\mathcal{M}$ transformation enriches the Class Activation Map by adding the dimension position information. Note that if we change the

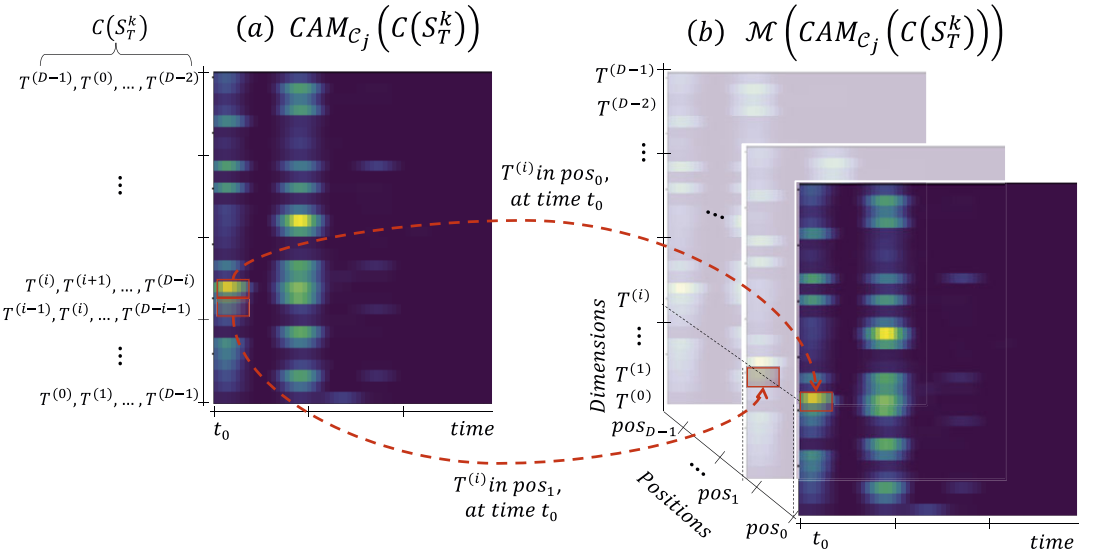

Figure 4. *Transformation $\mathcal{M}$ for a given data series T.*

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

dimension order of T, their $\mathcal{M}(\text{CAM}_{\mathcal{C}_j}(C(T)))$ changes as well. Indeed, for a given dimension $T^{(i)}$ and position p_j, $\text{idx}(T^{(i)}, p_j,)$ will not have the same value for two different dimension orders of T. Thus, computing $\mathcal{M}(\text{CAM}_{\mathcal{C}_j}(C(T)))$ for different dimension orders of T will provide distinct information regarding the importance of a given position (subsequence) in a given dimension. We expect that subsequences (of a specific dimension) that discriminate one class from another will also be associated (most of the time) with a high value in the Class Activation Map.

3.4.2. Merging permutations

We compute $\mathcal{M}(\text{CAM}_{\mathcal{C}_j}(C(S_T)))$ for different $S_T \in \Sigma_T$. Note that the total number of permutations for high-dimensional data series $|\Sigma_T|$ is enormous. In practice, we only compute $\mathcal{M}$ for a randomly selected subset of Σ_T. We thus merge $k = |\Sigma_T|$ permutations S_T^k, by computing the averaged matrix $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ of all the $\mathcal{M}$ transformations of the permutations. Formally, $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ is defined as follows:

$$\bar{\mathcal{M}}_{\mathcal{C}_j}(T) = \frac{1}{|\Sigma_T|} \sum_{S_T^k \in \Sigma_T} \mathcal{M}(\text{CAM}_{\mathcal{C}_j}(C(S_T^k)))$$

Figure 5 illustrates the process of computing $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ from the set of permutations of T, Σ_T. $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ can be seen as a summarization of the importance of each dimension at each position in $C(T)$, for all the computed permutations. Figure 5b' (at the top of the figure) depicts $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_d$, which corresponds to the d^{th} row (i.e., the dotted box in Figure 5b) of $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$. Each row of $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_d$ corresponds to the average activation of dimension d (for each timestamp) when dimension d is in a given position in $C(T)$.

Note that all permutations of T are forwarded into the dCNN network without training it again. Thus, even though the permutations of T generate radically different inputs to the network, the network can still classify most of the instances correctly. For k permutations, we use n_g to denote the number of permutations the model has correctly classified.

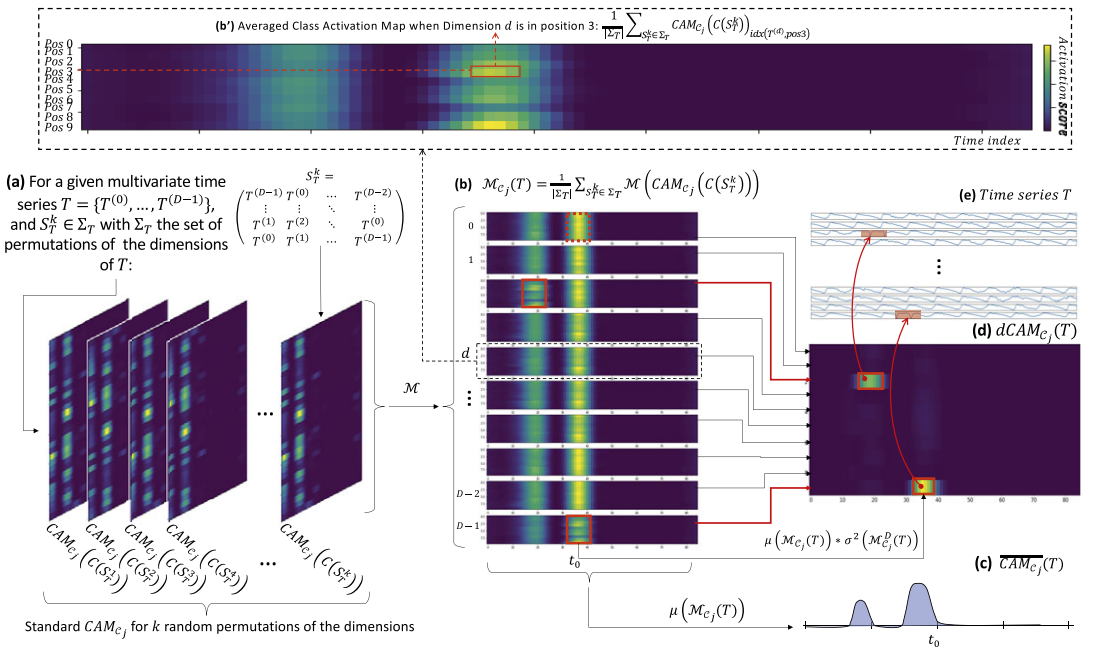

Figure 5. *dCAM computation framework.*

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

3.4.3. dCAM extraction

We can now use the previously computed $\bar{\mathcal{M}}_{\mathcal{C}_j}$ to extract explanatory information on which subsequences are considered important by the network. First, we note that each row of $C(T)$ corresponds to the input format of the standard CNN architecture. Thus, we expect that the result of a row of $\bar{\mathcal{M}}_{\mathcal{C}_j}$ (1 of the 10 lines in Figure 5b) is similar to the standard CAM. Hence, we can assume that $\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)\right) = \sum_{d\in[0,D-1]}\sum_{p\in[0,D-1]}\bar{\mathcal{M}}^{d,p}_{\mathcal{C}_j}(T)/(2*D)$ is equivalent to standard Class Activation Map $\mathrm{CAM}_{\mathcal{C}_j}(T)$ (this approximation is depicted in Figure 5d). Moreover, we can extract temporal information per dimension in addition to the global temporal information. We know that for a given position p and a given dimension d, $\bar{\mathcal{M}}^{d,p}_{\mathcal{C}_j}(T)$ represents the averaged activation for a given set of permutations. If the activation $\bar{\mathcal{M}}^{d,p}_{\mathcal{C}_j}(T)$ for a given dimension is constant (regardless of its value or the position p), then the position of dimension d is not important, and no subsequence in that dimension d is discriminant. On the other hand, a high or low value at a specific position p means that the subsequence at this specific position is discriminant. While it is intuitive to interpret a high value, interpreting a low value is counterintuitive. Usually, a subsequence at position p with a low value should be regarded as nondiscriminant. Nevertheless, if the activation is low for p and high for other positions, then the subsequence at position p is the consequence of the low value and is thus discriminant. We experimentally observe this situation, where a nondiscriminant dimension has a constant activation per position (e.g., see dotted red rectangle in Figure 5b: this pattern corresponds to a nondiscriminant subsequence of the dataset). On the contrary, for discriminant dimensions, we observe a strong variance for the activation per position: either high values or low values (e.g., see solid red rectangles in Figure 5b: these patterns correspond to the (injected) discriminant subsequences highlighted in red in Figure 5e). We thus can extract the significant subsequences per dimension by computing the variance of all positions of a given dimension. We can filter out the irrelevant temporal windows using the averaged $\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)\right)$ for all dimensions, and use the variance to identify the important dimensions in the relevant temporal windows. Formally, we define $\mathrm{dCAM}_{\mathcal{C}_j}(T)$ as follows.

Definition 3. For a given data series T and a given class $\mathcal{C}_i$, $\mathrm{dCAM}_{\mathcal{C}_j}(T)$ is defined as:

$$\mathrm{dCAM}_{\mathcal{C}_j}(T) = \begin{pmatrix} \sigma^2\left(\bar{\mathcal{M}}^{0}_{\mathcal{C}_j}(T)_{t_0}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}\right) & \dots & \sigma^2\left(\bar{\mathcal{M}}^{0}_{\mathcal{C}_j}(T)_{t_n}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}\right) \\ \vdots & \dots & \vdots \\ \sigma^2\left(\bar{\mathcal{M}}^{D-2}_{\mathcal{C}_j}(T)_{t_0}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}\right) & \dots & \sigma^2\left(\bar{\mathcal{M}}^{D-2}_{\mathcal{C}_j}(T)_{t_n}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}\right) \\ \sigma^2\left(\bar{\mathcal{M}}^{D-1}_{\mathcal{C}_j}(T)_{t_0}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}\right) & \dots & \sigma^2\left(\bar{\mathcal{M}}^{D-1}_{\mathcal{C}_j}(T)_{t_n}\right)*\mu\left(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}\right) \end{pmatrix} \quad (8)$$

3.5. Time complexity analysis

3.5.1. Training step

CNN/ResNet/InceptionTime require $O(\ell*|T|*D)$ computations per kernel, while dCNN/dResNet/dInceptionTime require $O\left(\ell*|T|*D^2\right)$ computations per kernel. Thus, the training time per epoch is higher for dCNN than CNN. However, given that the size of the input of dCNN is larger (containing D permutations of a single series) than CNN, the number of epochs to reach convergence is lower for dCNN when compared to CNN. Intuitively, dCNN trains on more data during a single epoch. This leads to similar overall training times.

3.5.2. dCAM step

The CAM computation complexity is $O\left(|T|*D*n_f\right)$, where n_f is the number of filters in the last convolutional layer. Let $N_f = \left[n_{f_1},\dots n_{f_n}\right]$ be the number of filters of the n convolutional layers. Then, a

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

forward pass has time complexity $O\left(\ell * |T| * D^2 * \sum_{n_{f_i} \in N_f} n_{f_i}\right)$. In dCAM, we evaluate k different permutations. Thus, the overall dCAM complexity is $O\left(k * \ell * |T| * D^2 * \sum_{n_{f_i} \in N_f} n_{f_i}\right)$. Observe that since the k permutations can be computed in parallel, the most important parameter for the execution time is D.

3.6. Further observations

3.6.1. Permutations success as a proxy

As previously explained, we assume that dCAM is meaningful if and only if the deep neural network classification is accurate. We also assume that classification accuracy impacts the number of correctly classified permutations. As in real use cases, labels may not be available, and both classification and discriminant feature identification accuracy may not be computed. Therefore, the number of correctly classified permutations (called n_g) could be used as a proxy to assess the quality of the explanation. More precisely, a low value of n_g indicates that the model is not efficient in classifying a time series regardless of the permutation. On the contrary, a high value of n_g indicates that, whatever the permutations of the time series dimensions, the model is able to classify correctly the time series. Therefore, as dCAM uses different permutations to find back which dimensions correspond to discriminant features, a low value of n_g indicates that the discriminant features identified by dCAM might not be meaningful.

3.6.2. From local to global explanations

We have so far assumed that dCAM is applied to a single multivariate data series and provides corresponding explanations. In the case where we need to analyze a set of series though, we can use dCAM on each one independently and then aggregate the dCAM results in order to identify global discriminant features. (The problem of local and global explanations has been discussed in other studies as well (Jacob et al., 2021)). Section 5 provides an example of how to get a global explanation of a specific use case. In future work, we will study other possibilities to merge dCAMs of instances of the same class.

3.6.3. Limitations: the dimension order problem

Even though this method is able to detect important subsequences within a multivariate data series, the use of dimension permutations can be problematic for specific time series. Such architecture is based on the use of dimension permutations as a fundamental principle. Thus, if the discriminant factor between two classes remains on the positions of the dimensions (e.g., having the same multivariate data series but with dimensions at different positions between two instances of two classes), one cannot use our proposed approach. Such a scenario is plausible for multivariate time series measured from a graph of sensors where the position of each sensor with regards to the other matters. In such cases, the differences between two events (i.e., two classes) might be explained by the geographical position of a specific pattern, resulting in discriminant features based on the order of the dimensions.

3.6.4. Limitations: large number of dimensions

Our proposed approach uses a new data structure $C(T)$. However, $C(T)$ has a memory complexity of $O(D^2)$. Such memory complexity can be problematic for specific time series with many dimensions. Such large time series would imply reducing the batch size during the training phase such that an entire batch fits the memory of a CPU or GPU. As many data points in $C(T)$ are redundant, an interesting research direction is to optimize the memory complexity to $O(D)$.

3.6.5. Limitations: CAM-related constraints

As mentioned in Section 2.2, the Class Activation Map can only be used if (1) a Global Average Pooling layer has been used before the softmax classifier and (2) the model accuracy is high enough.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

Consequently, only the standard architectures CNN and ResNet proposed in the literature, combined with a global average pooling layer, can benefit from CAM. Therefore, other layers or combinations of the convolutional layers outputs (such as flattening operations) limit the application and the usage of CAM. As dCAM is based on CAM, the above constraints also hold for dCAM.

4. Classification experimental evaluation

We now present the results of the experimental evaluation with several real datasets from different domains. Our code and datasets are available online (dCAM, 2022).

4.1. Experimental setup

We implemented our algorithms in Python 3.5 using PyTorch (Paszke et al., 2019). The evaluation was conducted on a server with Intel Core i7-8750H CPU 2.20GHz * 12, with 31.3GB RAM, and Quadro P1000/PCIe/SSE2 GPU with 4.2GB RAM.

4.1.1. Datasets

We conduct our experimental evaluation using real datasets injected with known discriminant patterns. We use the StarLightCurves (classes 2 and 3 only) and ShapesAll (classes 1 and 2 only) datasets from the UCR archive (Dau et al., 2019), in which we inject subsequences that will generate discriminant features. We build two types of datasets to study the ability of the algorithms to identify the discriminant patterns guiding the classification decision, (1) when these patterns occur in a subset of the dimensions at different timestamps, and (2) when these patterns occur in a subset of the dimensions at the same timestamp.

(1) For Type 1 datasets, we build each dimension of Class 1 by concatenating random instances from one class of one of our two UCR seed datasets. We build Class 2 by injecting in the series of the other class of our two UCR datasets a pattern in 2 random dimensions at a random position in the series.
(2) For Type 2 datasets, we build each dimension of Class 1 by concatenating random instances from one of the classes of our two UCR datasets and injecting patterns from the other class in x random dimensions and at different positions. We build Class 2 by injecting patterns at the same positions of 2 random dimensions.

Examples of Type 1 and Type 2 5-dimensional datasets based on StarLightCurves are depicted in Figure 6a,b, respectively. In our experiments, we generate 1000 time series for the Type 1 synthetic dataset and 1000 time series for the Type 2 synthetic dataset.

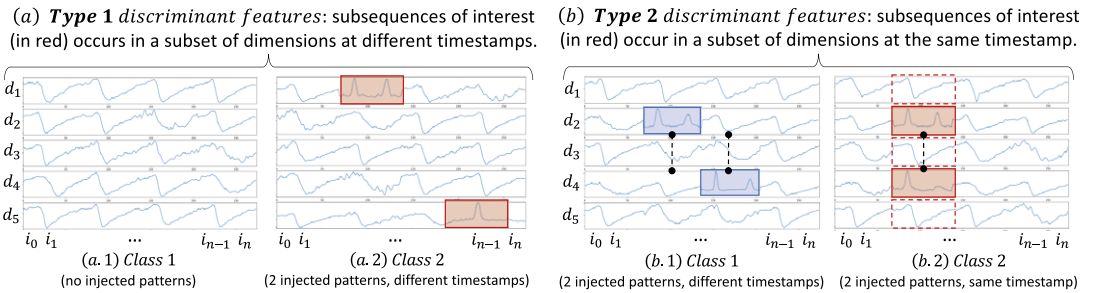

Figure 6. *Synthetic datasets: (a) Type 1, in which the discriminant subsequence is two injected patterns from class 2 StarLightCurves dataset in random dimensions at random positions, (b) Type 2, in which the discriminant factor is the fact that the two injected patterns are injected at the same position.*

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

4.1.2. Evaluation measures

We first evaluate the classification accuracy, C-acc. This measure corresponds to the ratio of correctly classified instances among all instances in the test dataset. We then evaluate the discriminant features accuracy, Dr-acc, for Class 1 (see Figure 6). We define Dr-acc as the PR-AUC for CAM/cCAM/dCAM obtained from the models and the ground-truth. The ground-truth is a series that has 1 at the positions of discriminant features (see Figure 6a.2): ground-truth contains 1 at the positions of the injected patterns, marked with the red rectangles, and 0 otherwise). We motivate the choice of PR-AUC (instead of ROC-AUC) because we are more interested in measuring the accuracy of identifying the injected patterns (representing at max 0.02 percent of the dataset) than measuring the accuracy of not detecting the noninjected patterns. In this very unbalanced case, PR-AUC is more appropriate than ROC AUC (Davis and Goadrich, 2006). Note that even though we annotate each point of the injected subsequences as discriminant, only some subparts of these sequences may be discriminant, thus leading to Dr-acc less than 1. Finally, for CNN/ResNet/InceptionTime we compute the Dr-acc scores by assuming that their (univariate) CAM values are the same for all dimensions. We mark their Dr-acc scores with a star in Table 1.

4.1.3. Architectures and training

We compare our model, dCNN/dResNet/dInceptionTime, to the classical ResNet model (Wang et al., 2017; Ismail Fawaz et al., 2018; Fawaz et al., 2019; Ismail Fawaz et al., 2020), and the cResNet baseline we introduced in Section 2. We only consider ResNet and cResNet architectures as we empirically observed that the latter are more accurate than CNN/cCNN and InceptionTime/cInceptionTime architectures. We are using the same architecture setup for all models. We then use CAM for ResNet, cCAM for cResNet, and dCAM for dCNN, dResNet, and dInceptionTime to identify discriminant features. For dCNN, we are using 5 convolutional layers with $(64,128,256,256,256)$ filters, respectively. We are using a kernel size of 3 and a padding of 2. For ResNet, cResNet and dResNet we use three blocks with three convolutional layers of 64 filters (for the first two blocks) and 128 layers (for the last block). We are using kernel sizes equal to 8, 5, and 3 for each block for the three layers of the block. For dInceptionTime, we use the same architecture as originally defined (Ismail Fawaz et al., 2020).

We also include MTEX-CNN (Assaf et al., 2019)(MTEX) as a baseline, representative of other kinds of architectures that can provide a multivariate CAM. The explanation is computed separately for discriminant features and timestamps using grad-CAM (Selvaraju et al., 2017) (MTEX-grad). The latter is a variant of the usual CAM.

We split our dataset into training and validation sets with 80 and 20 percents of the total dataset, respectively (equally balanced between the two classes). The training dataset is used to train the model, and the validation dataset is used as a validation dataset during the training phase. We generate a fully new test dataset of 1000 time series (generated in the same manner as the initial 1000 time series used for the train and validation set) and evaluate C-acc and Dr-acc. We train all models with a learning rate $\alpha = 0.00001$, a maximum batch size of 16 instances (less if GPU memory cannot fit 16 instances), and a maximal number of epochs equal to 1000 (we use early stopping and stop before 1000 epochs if the model starts overfitting the test dataset). For dCAM, we use $k = 100$ (number of random permutations), a value that we empirically verified (due to lack of space, a detailed analysis of the effect of k is in the full version of the article).

4.2. Accuracy evaluation

We now evaluate the classification accuracy (C-acc) and the discriminant feature identification accuracy (Dr-acc) on synthetically built datasets. Table 1 depicts both C-acc and Dr-acc on Type 1 and 2 datasets when varying the number of dimensions from 10 to 100.

Overall, we observe that all methods have better performance (both C-acc and Dr-acc) on Type 1 datasets than on Type 2 datasets. This was expected since discriminant features located in single dimensions are easier to find than discriminant features that depend on several dimensions.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

Table 1. C -acc *and* Dr-acc *averaged accuracy for 10 runs for MTEX-CNN, ResNet, cResNet, dCNN, dResNet and dInceptionTime over synthetic datasets*

Datasets			C-acc (averaged on 10 runs)					
Dataset	Type	Dim.	MTEX	ResNet	cResNet	dCNN	dResNet	dIncept.
StarLightCurves	Type 1	10	0.99 (±0.00)	0.95 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
		20	0.99 (±0.01)	0.71 (±0.03)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	0.98 (±0.01)
		40	0.98 (±0.05)	0.60 (±0.03)	1.00 (±0.00)	0.99 (±0.00)	1.00 (±0.00)	0.93 (±0.01)
		60	0.61 (±0.06)	0.57 (±0.01)	1.00 (±0.00)	0.98 (±0.00)	0.99 (±0.00)	0.91 (±0.01)
		100	0.55 (±0.04)	0.64 (±0.02)	1.00 (±0.00)	0.96 (±0.01)	0.97 (±0.00)	0.79 (±0.02)
	Type 2	10	0.58 (±0.04)	0.71 (±0.03)	0.53 (±0.02)	1.00 (±0.00)	1.00 (±0.00)	0.93 (±0.01)
		20	0.55 (±0.03)	0.61 (±0.02)	0.55 (±0.01)	0.98 (±0.00)	1.00 (±0.00)	0.70 (±0.03)
		40	0.56 (±0.03)	0.58 (±0.02)	0.51 (±0.01)	0.88 (±0.05)	0.58 (±0.02)	0.56 (±0.01)
		60	0.53 (±0.02)	0.55 (±0.04)	0.53 (±0.01)	0.64 (±0.08)	0.59 (±0.01)	0.55 (±0.02)
		100	0.52 (±0.01)	0.59 (±0.03)	0.50 (±0.01)	0.59 (±0.01)	0.56 (±0.01)	0.60 (±0.01)
ShapesAll	Type 1	10	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
		20	1.00 (±0.00)	0.86 (±0.02)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	0.99 (±0.00)
		40	0.85 (±0.00)	0.65 (±0.01)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
		60	0.83 (±0.16)	0.65 (±0.01)	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)	0.96 (±0.00)
		100	0.70 (±0.02)	0.57 (±0.04)	1.00 (±0.00)	0.98 (±0.00)	1.00 (±0.00)	0.85 (±0.01)
	Type 2	10	0.60 (±0.01)	0.82 (±0.03)	0.54 (±0.01)	1.00 (±0.00)	1.00 (±0.00)	0.93 (±0.01)
		20	0.54 (±0.02)	0.57 (±0.02)	0.52 (±0.02)	1.00 (±0.00)	1.00 (±0.00)	0.89 (±0.03)
		40	0.59 (±0.05)	0.60 (±0.03)	0.52 (±0.00)	0.90 (±0.03)	0.72 (±0.08)	0.73 (±0.10)
		60	0.57 (±0.03)	0.59 (±0.01)	0.51 (±0.00)	0.65 (±0.04)	0.61 (±0.01)	0.72 (±0.05)
		100	0.52 (±0.03)	0.59 (±0.02)	0.50 (±0.01)	0.55 (±0.01)	0.58 (±0.01)	0.55 (±0.02)
Rank			3.95	3.9	3	1.65	**1.6**	2.85

Datasets			Dr-acc (averaged on 50 instances)					
Dataset	Type	Dim.	MTEX	ResNet	cResNet	dCNN	dResNet	dIncept.
			MTEX-grad	CAM	cCAM		dCAM	
StarLightCurves	Type 1	10	0.40 (±0.09)	0.07 (±0.02)*	0.92 (±0.09)	0.46 (±0.08)	0.38 (±0.12)	0.21 (±0.24)
		20	0.38 (±0.00)	0.02 (±0.01)*	0.92 (±0.05)	0.38 (±0.03)	0.45 (±0.10)	0.36 (±0.19)
		40	0.24 (±0.03)	0.008 (±0.00)*	0.94 (±0.03)	0.28 (±0.05)	0.42 (±0.08)	0.39 (±0.14)
		60	0.05 (±0.09)	0.004 (±0.00)*	0.92 (±0.07)	0.23 (±0.05)	0.24 (±0.07)	0.13 (±0.06)
		100	0.01 (±0.08)	0.003 (±0.00)*	0.92 (±0.04)	0.20 (±0.06)	0.26 (±0.10)	0.10 (±0.03)
	Type 2	10	0.15 (±0.09)	0.0256 (±0.02)*	0.025 (±0.01)	0.26 (±0.07)	0.43 (±0.09)	0.10 (±0.07)
		20	0.04 (±0.04)	0.016 (±0.01)*	0.01 (±0.05)	0.28 (±0.06)	0.43 (±0.09)	0.05 (±0.02)
		40	0.07 (±0.08)	0.0068 (±0.00)*	0.006 (±0.01)	0.20 (±0.07)	0.05 (±0.05)	0.03 (±0.01)
		60	0.008 (±0.06)	0.0058 (±0.00)*	0.005 (±0.00)	0.01 (±0.00)	0.003 (±0.00)	0.009 (±0.01)
		100	0.01 (±0.10)	0.0024 (±0.00)*	0.002 (±0.02)	0.003 (±0.00)	0.004 (±0.01)	0.02 (±0.02)
ShapesAll	Type 1	10	0.60 (±0.30)	0.09 (±0.01)*	0.79 (±0.12)	0.66 (±0.05)	0.70 (±0.10)	0.55 (±0.17)
		20	0.31 (±0.01)	0.03 (±0.01)*	0.74 (±0.10)	0.56 (±0.06)	0.66 (±0.08)	0.51 (±0.20)
		40	0.20 (±0.23)	0.008 (±0.01)*	0.88 (±0.12)	0.45 (±0.06)	0.74 (±0.02)	0.76 (±0.19)
		60	0.50 (±0.02)	0.005 (±0.00)*	0.65 (±0.08)	0.44 (±0.05)	0.72 (±0.04)	0.79 (±0.20)
		100	0.002 (±0.02)	0.003 (±0.00)*	0.83 (±0.06)	0.31 (±0.09)	0.49 (±0.02)	0.48 (±0.23)
	Type 2	10	0.02 (±0.03)	0.0467 (±0.03)*	0.04 (±0.15)	0.63 (±0.10)	0.50 (±0.12)	0.32 (±0.21)
		20	0.04 (±0.02)	0.0132 (±0.02)*	0.013 (±0.15)	0.50 (±0.09)	0.73 (±0.08)	0.40 (±0.21)
		40	0.02 (±0.00)	0.005 (±0.00)*	0.005 (±0.00)	0.40 (±0.13)	0.20 (±0.14)	0.36 (±0.20)
		60	0.06 (±0.02)	0.0037 (±0.00)*	0.003 (±0.00)	0.22 (±0.13)	0.34 (±0.12)	0.46 (±0.17)
		100	0.04 (±0.02)	0.0027 (±0.00)*	0.002 (±0.00)	0.005 (±0.11)	0.02 (±0.05)	0.05 (±0.00)
Rank			3.85	4.45	3	2.6	**2.15**	2.75

We then notice that for low dimensional ($D = 10$) datasets, ResNet, dResNet, dCNN and dInceptionTime are performing nearly perfect C − acc. Moreover, ResNet and MTEX-CNN are performing well for low-dimensional data series but start to fail for a more significant number of dimensions. While the drop is already significant for the Type 1 dataset built from the StarLightCurve dataset, it is even stronger for Type 2 datasets, for which ResNet fails to classify instances with a number of dimensions $D \geq 20$. On the contrary, dCNN, dResNet, and dInceptionTime, which use the random permutations in the input, are not sensitive to the number of dimensions and have an almost perfect C-acc for most of Type 1 datasets. We observe a C-acc drop for dCNN, dResNet, and dInceptionTime as dimensions increase for Type 2 datasets. However, this drop is significantly less pronounced than that of ResNet. Overall, dCNN, dResNet, and dInceptionTime, which have on average the three highest ranks, are the most accurate methods.

Regarding cResNet, although it achieves a nearly perfect C-acc for Type 1 datasets, we observe that it fails to classify correctly instances of Type 2 datasets. As explained in Section 2, the input data structure is not rich enough to allow comparisons among dimensions, which is the main way to find discriminant features between the two classes of Type 2 datasets. We also observe that MTEX-CNN fails to classify instances of Type 2 datasets. Thus, this architecture is not correctly detecting the discriminant features across different dimensions. Overall, Figure 7a shows that dCNN, dResNet, and dInceptionTime are equivalent to cResNet for Type 1 (Figure 7a.1), outperforming all the baselines for Type 2 (Figure 7a.2), and in general are better than the baselines (ResNet and cResNet) for both types (Figure 7a.3) with $F(\text{Type 1}, \text{Type 2}) = \frac{2 * \text{C-acc(Type 1)} * \text{C-acc(Type 2)}}{\text{C-acc(Type 1)} + \text{C-acc(Type 2)}}$).

We now compare the different methods using the Dr-acc measure. We observe that the baseline cCAM (computed with cCNN) is outperforming CAM (computed with ResNet) and dCAM (with all of dCNN, dResNet, and dInceptionTime) for Type 1 datasets. This is explained by the fact that these classes can be discriminated by treating dimensions independently. Thus, cCAM (with no comparisons between dimensions) is naturally the best solution. Nevertheless, as Type 2 datasets require comparisons among dimensions to discriminate the classes, cCAM fails on them, with a Dr-acc very similar to the one of a random classifier. This confirms that such a baseline cannot be considered as a general solution for multivariate data series classification. We also observe that Dr-acc of the explanation method of MTEX-CNN (MTEX-grad) is lower than dCAM for Type 1 and close to Dr-acc of cCAM for Type 2, meaning that it cannot identify discriminant features of Type 2 datasets.

We then compare CAM and dCAM (used with dCNN, dResNet, and dInceptionTime). We note that dCAM significantly outperforms CAM. As depicted in Figure 7b, we also observe that Dr-acc reduces for

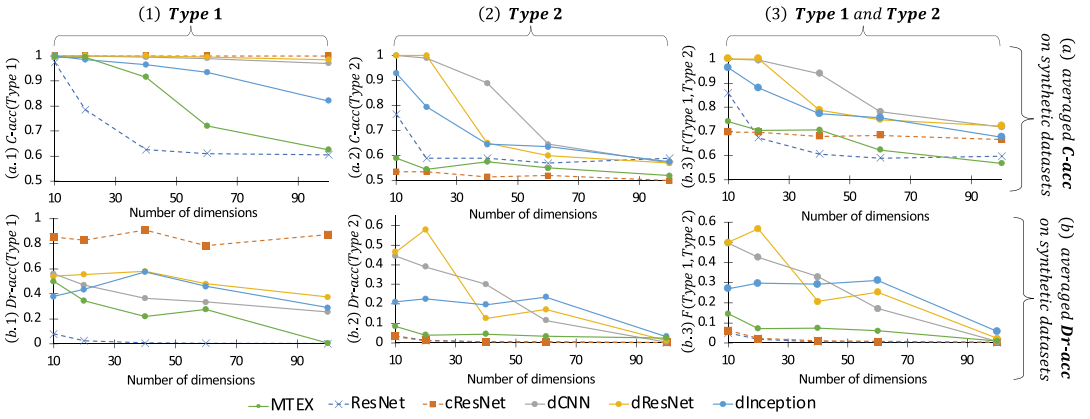

Figure 7. *Evaluation of the influence of the number of dimensions on our approaches and the baselines* C-acc *and* Dr-acc.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

all models as the number of dimensions increases. Nevertheless, Dr-acc of dCAM remains relatively high for both Type 1 (Figure 7b.1)) and Type 2 (Figure 7b.2) datasets (for a number of dimensions under 60).

This result demonstrates the superiority of dCAM over state-of-the-art methods. The superiority of dCAM is also confirmed by looking at the average ranks in Table 1, which indicates that dCAM computed from dResNet has the highest rank of 2.15.

5. Use case: precursors of anomalous vibration detection in nuclear power plants

We now illustrate the applicability and interest of our method in a real-world application. This use case is about discovering possible precursors of unwanted vibration happening in turbine-driven feed-water pump systems inside French nuclear power plants. These pumps (two different pumps noted, TPA1 and TPA2) aim to increase the water pressure (from 1 to 80 bar) before passing the water through the steam generator (with a pressure of 80 bar). However, these vibrations are problematic when the pump's position varies by a few microns, and a boolean sensor is activated when it happens. Thus, knowledge experts are interested in finding if there exist possible precursors of these unwanted vibrations in other sensors surrounding the pump and discovering unusual patterns that could explain why the pump is vibrating or at least alert the imminent occurrence of vibrations.

5.1. Dataset and use case description

At this point, we need to create our datasets of abnormal data series (i.e., vibrations) and anomaly-free data series. Following the suggestion of expert knowledge, we selected 120 sensors inside 12 subsystems of the nuclear power plants. Figure 8a summarizes the subsystems analyzed and the number of sensors collected. We collected every unwanted vibration that happened in every French 1300 MW nuclear plant. In total, we have 444 vibrations. We then create our multivariate data series by selecting every sensor's measurement between 75 minutes before the vibrations and 5 minutes after. We set the acquisition rate to 1 point every 6 seconds. Each multivariate data series contains 96,000 points. Next, we note the set of data series containing a vibration $T^A_{\mathcal{M}}$. We then select 444 intervals of 80 minutes for which no vibration has been recorded at least one day before and after. Finally, we note the set of data series without any vibration $T^N_{\mathcal{M}}$. We also selected the nonvibration periods to be under the same operating conditions as the vibration periods. Namely, when the nuclear facility ramps up or down between 15% and 67% of maximum power. This area, where the second feed-water pump is coupled, is conducive to vibrations. This is a critical step that must be conducted thoroughly. Otherwise, the precursors that would be highlighted would be the already-known difference in operational conditions and solicitations. In this specific case, we used the sensors related to the power regime: its distribution is the same across both classes. We also took the same distribution of years for both classes to minimize the influence of degradation due to aging. What we want to highlight are unexpected solicitations that would lead to vibrations later. This will trigger immediate and cost-effective corrective actions. We thus have in total 888 multivariate data series (for which 444 of them correspond to unwanted vibrations) of $D = 120$ dimensions. In total, the dataset contains 85,248,000 points. Formally, we define the dataset as $T_{\mathcal{M}} = T^N_{\mathcal{M}} \bigcup T^A_{\mathcal{M}}$, with $T^N_{\mathcal{M}}, T^A_{\mathcal{M}} \in \mathbb{R}^{(444,120,800)}$.

5.2. Experimental analysis

Overall, the task is to detect the vibration correctly and discover subsequences in one or several sensors that happened before the vibration and could potentially explain it. We tackle this task as defined in Problem 1 with dCNN/dCAM. Thus, we perform the following experiments. We first train dCNN to classify $T^A_{\mathcal{M}}$ and $T^N_{\mathcal{M}}$. Formally, dCNN is defined as a function $f : T^{\mathcal{N}}_{\mathcal{M}}, T^{\mathcal{A}}_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$. We then use dCAM as a function $g : T^{\mathcal{A}}_{\mathcal{M}}, f \rightarrow S$ that returns the set S of subsequences that explain the classification as the vibration class (as the red subsequences and rectangle depicted in Figure 8b,c). In practice, dCAM returns a multivariate data series score for each instance in $T^{\mathcal{A}}_{\mathcal{M}}$ (as depicted in Figure 8b,c). In Section 4.2,

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

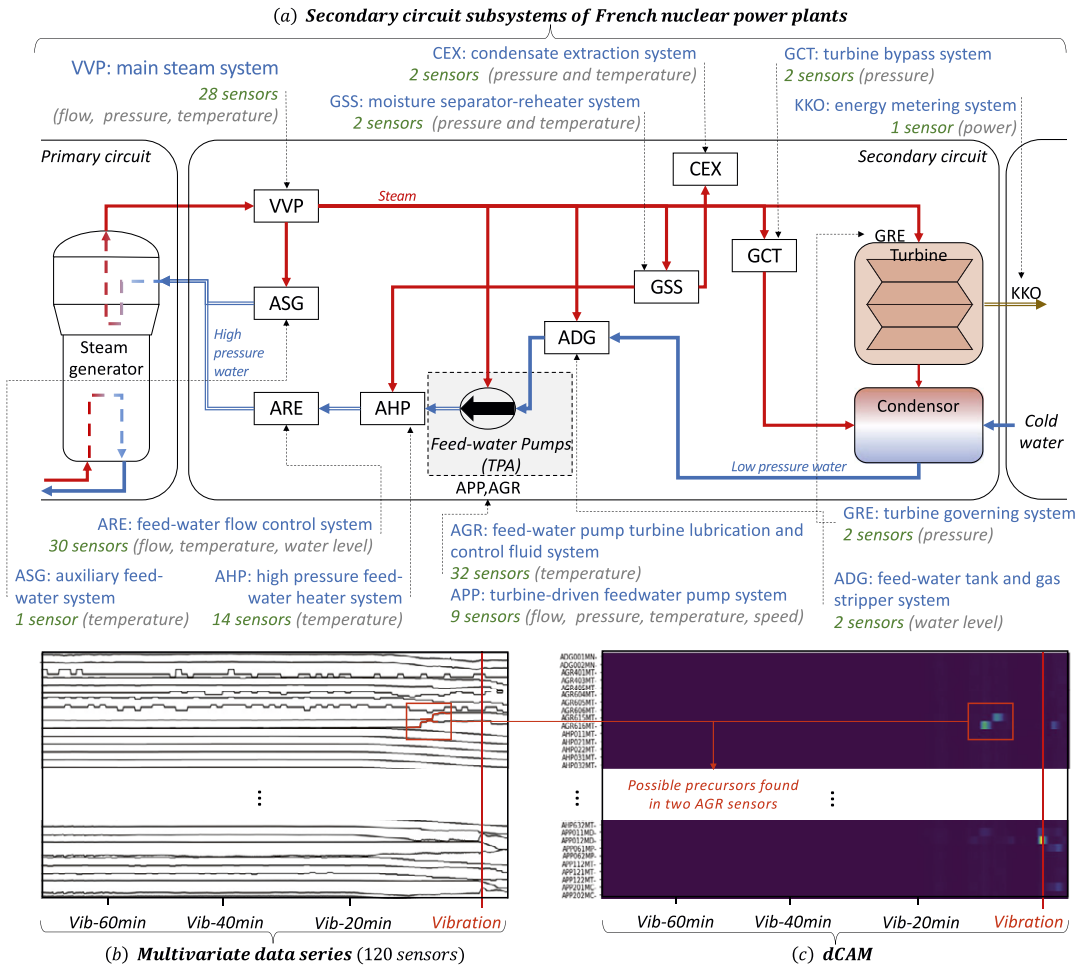

Figure 8. *Simplified scheme of the secondary circuit of 1300 MW nuclear power plant. We collect in total 120 sensors from 8 subsystems (solid black boxes) surrounding the feed-water pumps (TPA). Blue arrows: water flow. Red arrows: steam flows.*

we have shown that dResNet is slightly more accurate than dCNN and dInceptionTime. However, for simplicity, we used the most usual architecture dCNN as our baseline for this use case.

5.2.1. Accuracy evaluation

We train the dCNN (we are using 5 convolutional layers with (64,128,256,256,256) filters respectively; we are using a kernel size of 3 and a padding of 2) model on 70% of each class ($T^N_{\mathcal{M}}$ and $T^A_{\mathcal{M}}$) and we use the 30% left as a validation set. We set the batch size to 8 instances. We adopt an Early-stopping strategy to avoid overfitting. More precisely, we stop the training phase when the loss on the validation set is not reducing for the last 100 epochs. In total, we limit the training phase to 1000 epochs. It is important to note that the model and training parameters have been set empirically and based on the technical characteristics of our servers. More optimization with regards to the parameter selection could improve further the accuracy. However, the parameters listed above correspond to an adequate first baseline.

Overall, over 10 random splits between training and validation sets, dCNN achieved at best 0.91 accuracy on the training set and 0.89 accuracy on the validation set. As the dCNN accuracy is high, we can now use dCAM to identify the discriminant subsequences (i.e., possible vibration precursors).

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

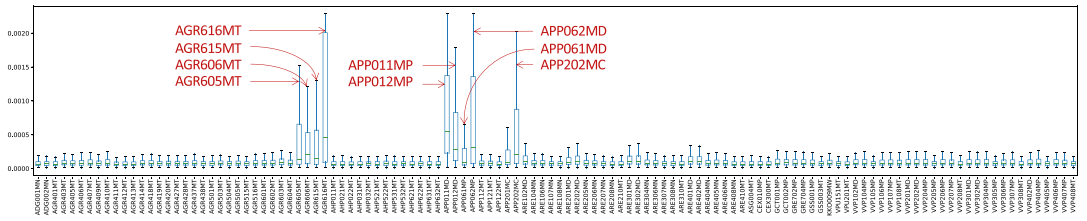

Figure 9. *Aggregated activation score for dCAM per sensor for every timestamp. In red: are the names of the sensors that are overall highly activated and possibly contain one or several precursors.*

5.2.2. Quantitative evaluation

We then evaluate the relevance of the subsequences identified by our proposed approach dCAM. We first start by measuring the consistency of the detection (or activation) as regards the (1) temporality (i.e., are the subsequences detected close in time to the vibration?), (2) structural information (i.e., are the sensors closely related to the vibrating system?). As the experts are interested in discriminant features across the entire dataset, we perform the analysis listed above on the entire $T^{\mathcal{A}}_{\mathcal{M}}$ (i.e., for time series included in the training and the validation set). As dCAM uses random permutations of the time series dimensions (resulting in inputs not used in the training set), the risks related to overfitting are limited and allow us to merge the training and validation set for this discriminant feature evaluation.

[Structural consistency] We then measure the average activation score per sensor for every timestamp. Figure 9 depicts the activation score box plot for each sensor using dCAM. We observe that the activation scores returned by dCAM vary significantly between different sensors. We can easily distinguish nine sensors out of 120 sensors. These sensors correspond to temperature measurements inside the feed-water pumps (sealing temperatures noted AGR605MT, AGR615MT for TPA1, and AGR606MT, AGR616MT for TPA2) and the outlet pump flow and pressures (noted APP011MD, APP061MP for TPA1, and APP012MD, APP062MP for TPA2). As highlighted in Figure 8a, AGR and APP are subsystems directly connected to the vibrating pump. Moreover, pressure and flow can directly influence the pump efficiency, with low-efficiency areas conducive to vibration events. Thus, the sensors highlighted by our proposed approach dCAM are very consistent with the knowledge from experts and the functional structure of the plant.

[Temporal consistency] We first measure the evolution of the average activation score (for all sensors) in time obtained by dCAM (Figure 10a). Figure 10 depicts the quantile values for each timestamp. The solid red line is the median, while every dotted gray line corresponds to the 20%-quantiles. We first observe that the average activation score is higher when the vibration occurs (red vertical line in Figure 10). As it is unlikely to find precursors one hour before the vibration, we can thus confirm that dCAM results are consistent regarding temporality. We then observe the average anomaly score for some specific sensors (highlighted in red in Figure 9) that are the most activated across all vibration instances. We observe that, on average, all these sensors see their activation increases approximately 10 minutes before the vibrations. We thus explore in the following section the activated subsequences for these nine sensors.

5.2.3. Qualitative evaluation

We now analyze in detail the results returned by dCAM and discuss the information that the knowledge experts can gain from it. We mainly focus our analysis on the nine most activated sensors (i.e., sensors depicted in Figure 10(2)(b)). We cluster (computed with the usual k-mean using Euclidean distance) the 15 minutes long subsequences with the highest activation score for a vibration instance. The centroids of these clusters thus represent the different shape categories within each sensor detected by dCAM.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

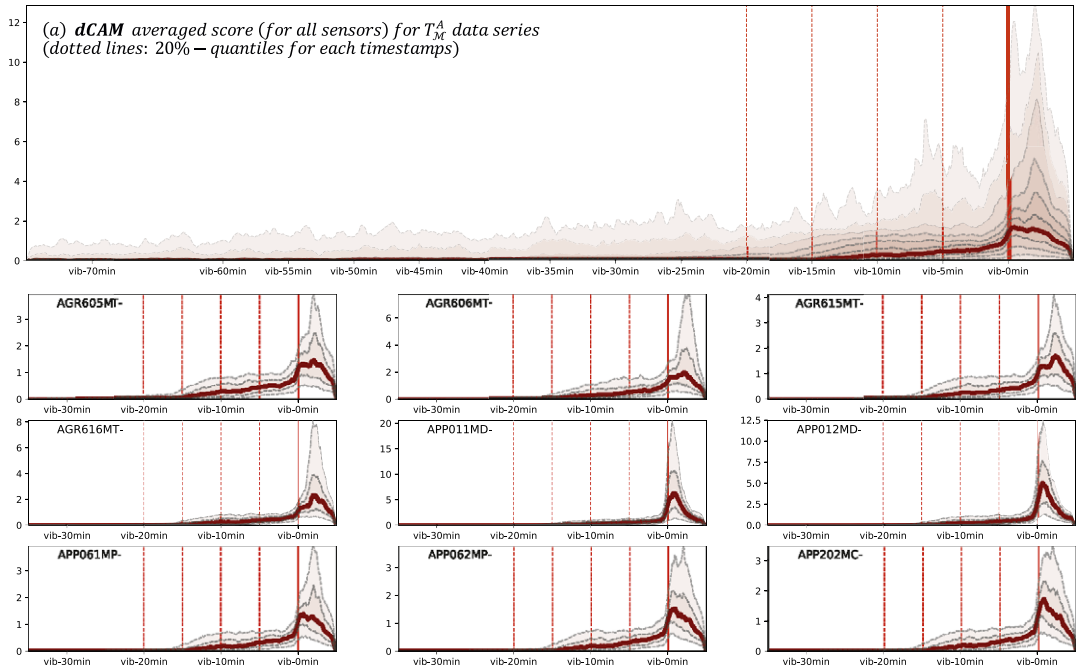

Figure 10. *Aggregated dCAM activation score for all sensors (a) and some specific sensors (b). Red shades correspond to quantile intervals (0.05–0.95,0.10–0.90,0.15–0.85, etc. for (a) and only 0.3–0.7, 0.4–0.6 for (b)). The solid red line is the median value for each timestamp.*

Therefore we can limit our analysis to this reduced (but relevant) set of centroids. Figure 11(2) depicts the 15 minutes long subsequences clusters in the nine sensors mentioned above. For each cluster, the time distribution histogram is displayed below. We first notice that the majority of the subsequences (for all clusters) happened while the vibration is detected (such as, for instance, the cluster depicted in Figure 11 (2)(b.2)). As understood by the experts, these subsequences correspond to a specific action (such as an increase or decrease of the water flow through the feed-water pump to either increase or decrease the power generated by the plant) that could lead to vibrations but that are not avoidable. Thus dCAM first permits the expert to confirm and visualize which subsequences are directly correlated to the vibration. Then, several subsequences detected by dCAM are anterior to the vibration (such as Figure 11(2)(a.2), (a.3),(b.1),(c.1),(c.2),(e.1),(g.3)). These subsequences could correspond to precursors of the vibration and would require to be carefully inspected by the experts. For instance, knowledge experts conclude that clusters such as in Figure 11(2)(e.1.1) correspond to unusual variations of the sealing temperature of the pump and lead them to investigate specific examples in detail. Overall, our proposed approach dCAM permits the experts to build a dictionary of patterns that can be related to a targeted anomaly. Thus the investigation by the expert could be significantly reduced.

Finally, the patterns depicted in Figure 11(2) can be connected based on their cooccurrence in time. As a result, Figure 11(1) displays a graph in which nodes and edges are defined as follows: nodes correspond to activated subsequences clusters (with part of them illustrated in Figure 11(2)), edges weights (proportional to the line width) correspond to the number of times a subsequence of one cluster happened at the same time as a subsequence in another cluster. Thus, in Figure 11(1), we observe two distinct communities of sensors. The right community corresponds mainly to sensors from the ARE and VVP subsystems. On the other hand, the most significant community (i.e., on the

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

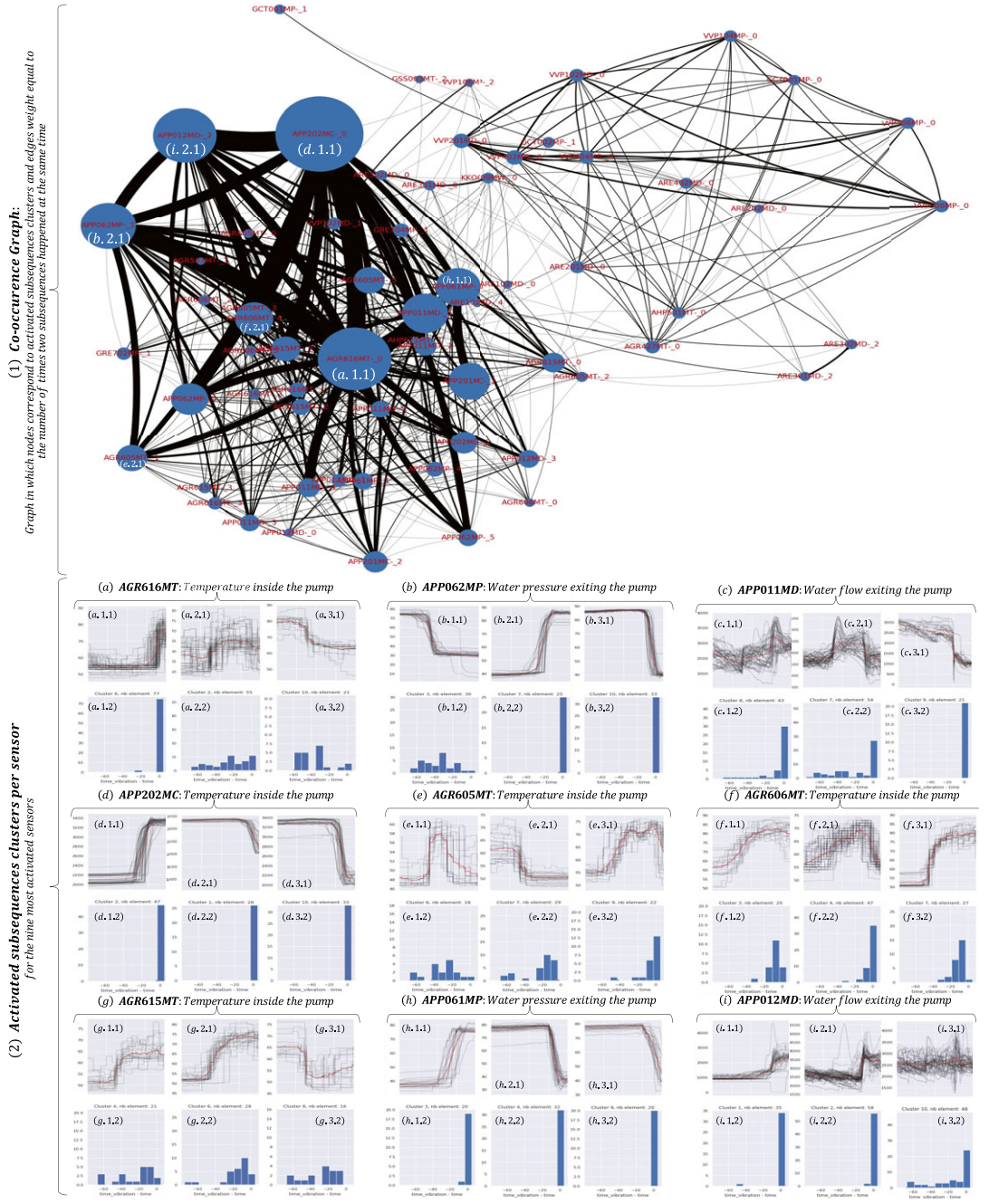

Figure 11. *(1) Cooccurrence graph connecting the activated subsequences clusters based on their cooccurrence in time. Subsequences clusters (*.*.1) (and their time distribution compared to the vibration timestamps (*.*.2)) detected as precursors of vibration by dCAM for the nine most activated sensors.*

left side of Figure 11(1)) are subsequences from the AGR and APP subsystems. These subsequences are mainly those illustrated in Figure 11(2). For instance, we learn from this graph that the patterns in Figure 11(2)(a.1.1), (d.1.1), (i.2.1), and (b.2.1) cooccurred very often before or during a vibration. Thus, such a graph can highlight more complex relationships between potential precursors.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

6. Conclusions and future work

Even though data series classification using deep learning has attracted a lot of attention, existing techniques for explaining classification decisions fail in the case of multivariate data series. In this work, we describe a novel approach, dCAM, based on convolutional neural networks, which enables us to detect the discriminant subsequences within individual dimensions of a multivariate data series. The experimental evaluation with synthetic and real datasets demonstrates the benefits and superiority of our approach in discriminant feature discovery and classification explanation in multivariate time series. Our real use case applied to the nuclear power plants domain verifies the applicability and the interest of our solution. In future work, we plan to study in detail the effect that using permutations in the input has on the overall approach.

Author contribution. Conceptualization: P.B., M.M., E.R.; T.P., B.D.; Data curation: P.B., M.M., B.D.; Data visualization: P.B., M.M.; Methodology: P.B., B.D.; Writing—original draft: P.B.; M.M., E.R., T.P.

Competing interest. P.B. was employed at EDF R&D and received grants from EDF R&D. M.M. and E.R. are employed at EDF R&D. B.D. is employed at EDF. T.P. received grants from EDF R&D.

Data availability statement. The data and code supporting this study's findings are openly available on GitHub. The code to reproduce the synthetic experiments can be found here: https://github.com/boniolp/dCAM. Restrictions apply to the availability of the data related to the industrial use case described in this article.

Funding statement. This work was supported by EDF R&D and ANRT French program.

Ethical standard. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

References

Assaf R, **Giurgiu I**, **Bagehorn F and Schumann A** (2019) Mtex-CNN: Multivariate time series explanations for predictions with convolutional neural networks. In *IEEE International Conference on Data Mining (ICDM)*, pp. 952–957.

Bagnall AJ, **Cole RL**, **Palpanas T and Zoumpatianos K** (2019) Data series management (dagstuhl seminar 19282), Dagstuhl Reports 9 (7e).

Bagnall A, **Lines J**, **Hills J and Bostrom A** (2015) Time-series classification with cote: The collective of transformation-based ensembles. *IEEE TKDE 27*, 2522.

Barnet V and Lewis T (1994) *Outliers in Statistical Data*. New York: John Wiley and Sons.

Boniol P, **Linardi M**, **Roncallo F**, **Palpanas T**, **Meftah M and Remy E** (2021) Unsupervised and scalable subsequence anomaly detection in large data series. *The VLDB Journal.*

Boniol P and Palpanas T (2020) Series2graph: Graph-based subsequence anomaly detection for time series, Proc. *VLDB Endow 13*, 11.

Boniol P, **Paparrizos J**, **Kang Y**, **Palpanas T**, **Tsay RS**, **Elmore AJ and Franklin MJ** (2022) Theseus: Navigating the labyrinth of subsequence anomaly detection, Proc. *VLDB Endow 15*, 12.

Boser BE, **Guyon IM and Vapnik VN** (1992) A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*. New York: Association for Computing Machinery, pp. 144–152.

Breunig MM, **Kriegel H-P**, **Ng RT and Sander J** (2000) *Lof: Identifying Density-Based Local Outliers*. Portland: OR: SIGMOD.

Chen H, **Tang F**, **Tino P and Yao X** (2013) *Model-Based Kernel for Efficient Time Series Analysis*. Long Beach, CA: ACM SIGKDD.

Christ M, **Braun N**, **Neuffer J and Kempa-Liehr AW** (2018) Time series feature extraction on basis of scalable hypothesis tests (tsfresh – A python package). *Neurocomputing 307*, 72–77.

Christ M, **Kempa-Liehr AW and Feindt M** (2016) Distributed and parallel time series feature extraction for industrial big data applications. arXiv preprint arXiv:1610.07717.

Cui Z, **Chen W and Chen Y** (2016) Multi-scale convolutional neural networks for time series classification, CoRR.

Dau HA, **Bagnall A**, **Kamgar K**, **Yeh CM**, **Zhu Y**, **Gharghabi S**, **Ratanamahatana CA and Keogh E** (2019) The ucr time series archive. *IEEE/CAA Journal of Automatic Sinica 6*, 6.

Davis J and Goadrich M (2006) The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, pp. 233–240.

dCAM (2022) Source code. Available at https://github.com/boniolp/dCAM.

Fawaz HI, **Forestier G**, **Weber J**, **Idoumghar L and Muller P-A** (2019) Deep learning for time series classification: A review, data min. *Knowledge Discovery 33*, 4.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

Freund Y and Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: *Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT '95*. Berlin: Springer-Verlag, pp. 23–37.

Hinton GE (1989) Connectionist learning procedures. *Artificial Intelligence 40*, 185–234.

Ho TK (1995) Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition 1*, pp. 278–282.

Hsieh T-Y, **Wang S**, **Sun Y and Honavar V** (2021) Explainable multivariate time series classification: A deep neural network which learns to attend to important variables as well as time intervals. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining, WSDM '21*, pp. 607–615.

Ismail Fawaz H, **Forestier G**, **Weber J**, **Idoumghar L and Muller P-A** (2018) *Evaluating Surgical Skills from Kinematic Data Using Convolutional Neural Networks*. Marrakesh: MICCAI.

Ismail Fawaz H, **Lucas B**, **Forestier G**, **Pelletier C**, **Schmidt D**, **Weber J**, **Webb G**, **Idoumghar L**, **Muller P and Petitjean F** (2020) Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery 34*, 1936–1962.

Ismail AA, **Gunady MK**, **Bravo HC and Feizi S** (2020) Benchmarking deep learning interpretability in time series predictions. NeurIPS 2020.

Jacob V, **Song F**, **Stiegler A**, **Rad B**, **Diao Y and Tatbul N** (2021) Exathlon: A benchmark for explainable anomaly detection over time series. *Proceedings of the VLDB Endowment 14*(11), 2613–2626.

Jakovljevic L, **Kostadinov D**, **Aghasaryan A and Palpanas T** (2022) Towards building a digital twin of complex system using causal modelling. In Benito RM, Cherifi C, Cherifi H, Moro E, Rocha LM and Sales-Pardo M (eds), *Complex Networks & Their Applications X*. Cham: Springer International Publishing, pp. 475–486.

Keogh E, **Lonardi S**, **Ratanamahatana CA**, **Wei L**, **Lee S-H and Handley J** (2007) Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery.*

Kingma DP and Ba J (2015) *Adam: A Method for Stochastic Optimization*. Kigali: ICLR.

Krizhevsky A, **Sutskever I and Hinton GE** (2012) Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 84.

Le Guennec A, **Malinowski S and Tavenard R** (2016) Data augmentation for time series classification using convolutional neural networks. In: *ECML/PKDD on AALTD Workshop.*

LeCun Y, **Bengio Y and Hinton G** (2015) Deep learning. *Nature 521*, 436.

Lines J, **Taylor S and Bagnall A** (2016) *Hive-Cote: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification*. Auckland: IEEE ICDM.

Liu Y, **Chen X and Wang F** (2009) Efficient detection of discords for time series stream. In *Advances in Data and Web Management*. Berlin: Springer, pp. 629–634.

Liu FT, **Ting KM and Zhou Z-H** (2008) Isolation forest. In: *ICDM.*

Lundberg SM and Lee S-I (2017) A unified approach to interpreting model predictions. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*. Red Hook, NY: Curran Associates, pp. 4768–4777.

Ma H, **Ghojogh B**, **Samad MN**, **Zheng D and Crowley M** (2020) Isolation mondrian forest for batch and online anomaly detection.

Nair V and Hinton GE (2010) *Rectified Linear Units Improve Restricted Boltzmann Machines*. Baltimore, MD: ICML.

Palpanas T (2015) Data series management: The road to big sequence analytics. *SIGMOD Record 44*(2), 47–52.

Paparrizos J, **Boniol P**, **Palpanas T**, **Tsay RS**, **Elmore AJ and Franklin MJ** (2022a) Volume under the surface: A new accuracy evaluation measure for time-series anomaly detection. *Proceedings of the VLDB Endowment.*

Paparrizos J, **Kang Y**, **Boniol P**, **Tsay R**, **Palpanas T and Franklin MJ** (2022b) TSB-UAD: An end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment 15*(8), 1697–1711.

Paszke A, **Gross S**, **Massa F**, **Lerer A**, **Bradbury J**, **Chanan G**, **Killeen T**, **Lin Z**, **Gimelshein N**, **Antiga L**, **Desmaison A**, **Kopf A**, **Yang E**, **DeVito Z**, **Raison M**, **Tejani A**, **Chilamkurthy S**, **Steiner B**, **Fang L**, **Bai J and Chintala S** (2019) Pytorch: An imperative style, high-performance deep learning library. *NeurIPS 32*

Ribeiro MT, **Singh S and Guestrin C** (2016) "Why should I trust you?": Explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*. New York, NY: Association for Computing Machinery, pp. 1135–1144.

Selvaraju RR, **Cogswell M**, **Das A**, **Vedantam R**, **Parikh D and Batra D** (2017) Grad-cam: Visual explanations from deep networks via gradient-based localization. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626.

Senin P, **Lin J**, **Wang X**, **Oates T**, **Gandhi S**, **Boedihardjo AP**, **Chen C and Frankenstein S** (2015) *Time Series Anomaly Discovery with Grammar-Based Compression*. New York: EDBT.

Serrà J, **Pascual S and Karatzoglou A** (2018) *Towards a Universal Neural Network Encoder for Time Series*. Washington, DC: CCIA.

Subramaniam S, **Palpanas T**, **Papadopoulos D**, **Kalogeraki V and Gunopulos D** (2006) Online outlier detection in sensor data using non-parametric models. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*.

Wang Q and Palpanas T (2021) *Deep Learning Embeddings for Data Series Similarity Search*. San Diego, CA: SIGKDD.

Wang J, **Wang Z**, **Li J and Wu J** (2018) Multilevel wavelet decomposition network for interpretable time series analysis. ACM SIGKDD.

https://doi.org/10.1017/dce.2023.25 Published online by Cambridge University Press

Wang Z, **Yan W and Oates T** (2017) *Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline*. IJCNN.

Xu B, **Wang N**, **Chen T and Li M** (2015) Empirical evaluation of rectified activations in convolutional network. In *Deep Learning Workshop ICML.*

Yeh, **Zhu Y**, **Ulanova L**, **Begum N**, **Ding Y**, **Dau HA**, **Silva DF**, **Mueen A and Keogh EJ** (2016) *Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets*. ICDM, pp. 1317–1322.

Zhang H (2004) *The Optimality of Naive Bayes*. Miramar Beach: The Florida AI Research Society.

Zhao B, **Lu H**, **Chen S**, **Liu J and Wu D** (2017) Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics 28*, 1.

Zheng Y, **Liu Q**, **Chen E**, **Ge Y and Zhao JL** (2014) *Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks*. WAIM.

Zhou B, **Khosla A**, **Oliva LAA and Torralba A** (2016) Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, pp. 2921–2929.

Cite this article: Boniol P, Meftah M, Remy E, Didier B and Palpanas T (2023). dCNN/dCAM: anomaly precursors discovery in multivariate time series with deep convolutional neural networks. *Data-Centric Engineering*, 4, e30. doi:10.1017/dce.2023.25