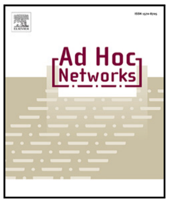
Ad Hoc Networks 156 (2024) 103412

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc



Asymmetric Autoencoders: An NN alternative for resource-constrained devices in IoT networks

Mateus S. Gilbert [a,*], Marcello L.R. de Campos [b], Miguel Elias M. Campista [a]

[a] *Grupo de Teleinformática e Automação – GTA, PEE/COPPE-DEL/POLI, Universidade Federal do Rio de Janeiro – UFRJ, Rio de Janeiro, RJ, Brazil*
[b] *Laboratório de Sinais, Multimídea e Telecomunicações – SMT, PEE/COPPE-DEL/POLI, Universidade Federal do Rio de Janeiro – UFRJ, Rio de Janeiro, RJ, Brazil*

A R T I C L E   I N F O

Keywords:
Internet of Things
Wireless sensor networks
Neural networks
Temporal compression
Data denoising
Asymmetric autoencoders

A B S T R A C T

Local computation and communication are known challenges for energy-constrained devices that can become even more complex if we consider data acquisition with noise. Thus, developing systems that address these problems is fundamental when implementing sensing nodes in IoT networks. Fortunately, sensed data has intrinsic redundancies that allow compression with little or no information loss, which can even be used to suppress the collected noise. Many solutions using Neural Networks (NNs) have emerged to address both issues, resorting to autoencoders to extract these redundancies to reduce data transmissions in IoT networks and to remove noise from data in general. However, solutions that resort to NNs often rely on increasing the number of NN layers to achieve performance improvements, which can be tricky when deploying them in resource-constrained devices. Models with multiple layers require more space to store their parameters and more computations. To address these problems, we propose Asymmetric Autoencoders (AAEs), a model that modifies the typical autoencoder, which adopts a symmetric encoder-decoder architecture, in favour of a design that has fewer NN and other resources in the encoder than in the decoder. Our experiments with single-sensor temporal-data compression show that our proposed AAEs can offer a similar or smaller reconstruction error compared to the symmetric AEs while using encoders with fewer parameters and that require fewer floating-point operations (FLOPs) with each compression operation. For instance, the proposed AAEs can outperform the best symmetrical implementations by executing five to seven times fewer FLOPs. Given their inherently IoT-friendly design and positive results, we show that AAEs are a valuable model for NN deployment in sensor nodes, as they can achieve similar or better performance than symmetric autoencoders while saving sensor node resources.

1. Introduction

The Internet of Things (IoT) paradigm integrates communication capacity to electronic devices, exploiting the already available Internet infrastructure. A typical IoT network comprises sensors and actuators capable of communicating with each other and the Internet [1]. Hence, electronic devices ranging from all-purpose sensing nodes to smart devices, e.g., smartphones and self-driving cars, become components of the IoT network environment.

Crucial for many IoT applications is data collection. However, a common problem for sensor devices is energy efficiency [2,3], which can make data collection particularly challenging. Devices are typically powered by rechargeable batteries, imposing issues regarding power consumption and quick energy depletion. Hence, applications that exhibit energy-efficient behaviour are welcome, as they save the trouble of constantly recharging batteries or even sending people to inhospitable places for battery replacement. In IoT, a well-known source of energy consumption is wireless transmissions, compelling many applications to focus on performing this task efficiently. Because IoT networks are expected to contain multiple sensing nodes collecting data from distinct sources, the development of applications that can work with high volumes of data to minimize information size transferred is crucial for saving data transmission bandwidth [4]. Since early studies in Wireless Sensor Networks (WSNs), a common practice is to avoid redundant data transmissions, even at the expense of local computation [2]. For instance, sensing nodes monitoring the same phenomena accumulate measurements that carry space–time correlation, which can be exploited to reduce the amount of data sent from IoT sensor nodes [3]. Hence, compression algorithms or general data aggregation approaches can reduce the amount of data transmitted by exploiting data redundancy [5].

Besides energy savings, noise poses a significant challenge to IoT applications. This problem is particularly pronounced in low-cost devices

* Corresponding author.
E-mail addresses: gilbert@gta.ufrj.br (M.S. Gilbert), campos@smt.ufrj.br (M.L.R. de Campos), miguel@gta.ufrj.br (M.E.M. Campista).

https://doi.org/10.1016/j.adhoc.2024.103412
Received 28 September 2023; Received in revised form 20 December 2023; Accepted 18 January 2024
Available online 7 February 2024

with limited resources [6]. Additionally, the deployment environment and the network density affect the collected noise intensity. Natural changes in the surroundings can affect the perceived noise, and so can neighbouring devices, whose operations can interfere with one another. Therefore, incorporating noise suppression as an additional feature of the compression system can significantly enhance the effectiveness of IoT data collection.

With the recent popularization of deep learning, new compression schemes for WSN (and consequently for IoT devices) have been proposed using neural networks (NNs) [7–10]. Many phenomena monitored in IoT settings require nonlinear models, where NNs suit well. NNs can adapt to data distribution, allowing the construction of compression systems that operate with little prior knowledge of the monitored signal. Also, NNs can impose a relatively lightweight computational cost, as the encoder stored in the sensor node can be the resulting weight matrix with an activation function (to perform the nonlinear transformation) [7]. However, many solutions that resort to NNs usually rely on increasing the number of layers and other computational resources to improve performance. This approach can be problematic when deploying NNs in IoT networks, as implementing more complex models on resource-constrained devices can be more difficult. Moreover, it will be common for sensing nodes to require multiple NN models to deal with heterogeneous data [3], meaning that each model's requirements can stack up and become a problem for IoT nodes. We address this issue by relying on a relevant recommendation for designing compression and aggregation strategies for IoT networks, which is usually neglected or an afterthought on typical NN architectures: the development of asymmetric schemes. As pointed out in early discussions in compression schemes for WSNs, the bulk of computation should be performed at the decoder rather than at the sensor node [11]. The current literature often uses compression schemes based on NNs that evenly distribute the compression and decompression tasks between the sensor and the sink [6,7]. Hence, this paper investigates new approaches that seek NN depth increase considering an asymmetrical design, filling an existing gap in the NN deployment in IoT literature.

This paper proposes Asymmetric Autoencoder (AAE) architectures for IoT data compression, in which the number of existing layers in the decoding block is greater than in the encoding part. With the proposed architecture, one can seek performance improvements by increasing the number of layers, and other NN features, without imposing additional resource demands on sensitive devices. We train our proposed AAEs to perform noise suppression and compression tasks simultaneously, and show the suitability of the proposed models for compressing IoT data under different noise levels. Additionally, we benefit from data temporal correlations to propose AAE architectures that deploy convolutional layers, referred to as Convolutional AAEs (CAAEs), to enhance the usage of short- and medium-term relations. With these proposed architectures, we show that one can add additional features to the decoding layers to exploit data characteristics further. Ultimately, the main contribution of our proposed models is enabling the development of compression systems that keep the storage and computational advantages found in the shallow architecture from previous papers [7], and achieve performance improvements by increasing the depth and modifying the decoding block of the autoencoder.

The results back our assumptions that AAEs fit well in resource-constrained devices, which are sensor nodes in typical IoT networks. We show that our proposed models can offer a similar or smaller reconstruction error when compared to the symmetric AEs using encoders with fewer parameters and layers, which require fewer floating-point operations (FLOPs) and storage space. Moreover, the pros of the AAEs go beyond an IoT-friendly architecture, handling some IoT dataset inconveniences, such as an insufficient number of training samples. Additionally, our results show that the trained models can outperform the symmetrical counterparts and achieve nearly identical reconstruction errors independent of the signal-to-noise ratio (SNR), varying from 80 db down to 20 db. Specifically, in the case of CAAEs, results suggest that incorporating computational structures that exploit data particularities is desirable, as they have achieved the best performance among all trained models. Overall, all these results highlight the main contribution of this paper: AAEs are a valuable model for autoencoder deployment in IoT networks, as their architecture enables the pursuit of performance gains without requiring a significant increase in sensor node cost.

The remainder of this paper is organized as follows. First, in Section 2, we overview the literature on data compression and denoising using autoencoders. Next, Section 3 provides background on data compression using autoencoders. Also, in this section, we discuss data denoising briefly. Section 4 proposes the AAEs, discussing the design choices for the asymmetrical architectures. In the following, Section 5 details the dataset selected and explains the procedure conducted for autoencoder training. Then, in Section 6, we present the obtained results. Finally, we conclude this paper in Section 7.

2. Related work

Usually, when resorting to NN-based solutions, autoencoders are the model of choice when dealing with dimensionality reduction and data enhancement problems. Hence, several works considering data compression and noise suppression for IoT networks use autoencoders [6]. Alsheik et al. demonstrate the capabilities of a compression scheme based on a shallow symmetric autoencoder architecture to deal with IoT data [7]. The authors employ the proposed model to handle temporal series, showing that their proposal can deliver a lightweight compression scheme that significantly reduces energy consumption. As expected, increasing the depth of this symmetric autoencoder yields better data compression results. These results occur because the more layers the neural network has, the more complex functions it learns. For instance, in scenarios related to dimensionality reduction, which shares similarities with data compression, this trend is also observed [10]. However, increasing the number of layers can be a problem, as more layers in the encoder portion incur storing more parameters and running more computations in the sensor. We present an asymmetrical autoencoder architecture as an alternative to this approach. In our proposal, the number of layers is the same as in Alsheik et al. but the decoder can have as many layers as possible. Thus, we show that it is possible to keep the benefits of a lightweight encoder at the sensor without affecting performance. In addition, our approach deals with data corrupted with noise.

Regarding noise suppression, two recent works use Denoising Autoencoders (DAEs) to remove noise from data. Laakom et al. [12] propose a new loss function to help the model learn how to remove the noise. Even though they consider data compression, they focus on denoising, showing that autoencoders can deal with both challenges. Additionally, their model is symmetrical and has multiple encoding layers, as it does not consider IoT data. In another work, Lee et al. [13] propose a modification to DAE, which shifts from the traditional approach of learning how to reconstruct the noiseless signal to extract the noise, subtracting it from the signal. They show that their approach outperforms DAE in the evaluated scenario, but differs from ours by resorting to a symmetrical architecture and not addressing data compression.

Table 1 summarizes the differences between our work and other proposals that involve data processing with autoencoders. Overall, our paper presents multiple AAEs models, showing that one can add multiple layers and features to the decoder to improve performance. Hence, our key contribution is presenting asymmetrical NN architectures that can be a valuable alternative when implementing applications that use autoencoders in IoT networks. Through our experiments, we show that the AAEs deliver comparable performances to their symmetrical counterparts, capable of even outperforming them. Hence, in addition to its architecture suited for resource-constrained devices, we show

Table 1
Summary of related work resorting to autoencoders for data compression (or dimensionality reduction) and noise removal.

Reference	Data compression & Dim. reduction	Noise removal	Single encoding layer	Asymmetric design
[7]	✗		✗	
[10]	✗		*partial*	
[12]	*partial*	✗		
[13]		✗		
[15]	✗			✗
This work	✗	✗	✗	✗

that AAEs can be fundamental to NN deployment in IoT. We refer to Mohammadi et al. [14] and Bochie et al. [6] for other cases where AEs are used, in which AAEs can be beneficial.

With the popularization of machine learning in recent years, especially thanks to the recent success of DNNs, a great effort is being spent in simplifying model requirements to embed them in resource-constrained devices. Tiny Machine Learning (TinyML) is a paradigm that emerged recently, focusing on designing ML models for low-power and low-cost microcontrollers [16]. Among different approaches, TinyML usually relies on parameter pruning and quantization to offer more compact and computationally efficient models [16,17]. Although crucial for implementing DNNs in resource-constrained devices, quantization can lead to performance degradations because 32-bit floating point parameters are mapped into representations that require fewer bits, thus losing representation precision. Additionally, the current best quantization techniques can offer a maximum size reduction of four times [17], and pruning-like techniques are more effective in over-parameterized models that have many infinitesimal parameters. This means that DNNs that are big to begin with naturally take more space than models that are designed with fewer layers and parameters. An IoT sensing node is expected to collect data from multiple sources [3], so it will likely store multiple ML models to treat each one individually or will require a complex model capable of dealing with this heterogeneous data. With our proposal, one can start with a more compact model, which can downplay the need for parameter quantization and pruning. Moreover, because our proposal focuses on the architectural aspect of autoencoders, it can be complementary to the aforementioned TinyML technics in designing DNNs that require less storage space and computations, as we can start with a model that is already compact and energetically efficient in comparison with a more complex symmetric autoencoder.

Asymmetric autoencoders were previously analysed [15] but with an encoder containing more layers than the decoder. This architecture is the opposite required in typical IoT scenarios, as the bulk of computation would be shifted towards the resource-constrained nodes. Ideally, in an IoT network, it is desirable to outsource most computation to a resource-rich central node. Hence, to our knowledge, our paper is the first to propose an asymmetrical architecture more suited for IoT deployments.

3. Autoencoders: General features and usage

Autoencoder networks (AE) are central to dimensionality reduction and data enhancement using NNs. AEs are structured to replicate the input data in its output layer, extracting relevant features throughout the intermediate layers. In the scenario analysed, the extracted features are crucial for generating a compact representation of the input data that allows reconstruction with noise suppression.

Fig. 1 illustrates a generic feedforward AE. In feedforward AEs, only neurons in adjacent layers interconnect, and data flows from the input to the output layers without loops. Another essential characteristic shown in Fig. 1 is that the AE is undercomplete, meaning that hidden layers have fewer neurons than the input and output layers [18]. Consequently, the AE architecture naturally forces the learning of a more compact representation, leading to data compression. The autoencoders used in this paper are of this type, with the term undercomplete omitted (which is usually the case). An autoencoder is typically split into two symmetrical parts for data compression (encoder) and decompression (decoder).

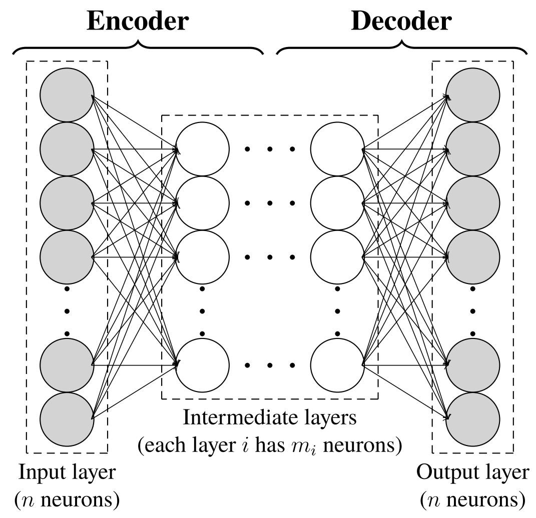


Fig. 1. Symmetric Autoencoder: input and output layers have n neurons, whereas each hidden layer i has m_i neurons, $m_i < n$. Also, the number of layers of the encoder and the decoder are the same.

Recent data compression efforts in IoT have found AEs particularly useful [6]. Because NNs fall under the representation learning paradigm, data compression requires little prior knowledge about intrinsic data characteristics. NNs learn on the fly the transformations needed to compress and recover the noiseless data [19]. Moreover, the presence of nonlinear activation functions at the neurons enables the models to adapt to a wide range of functions. Hence, AEs are an attractive tool for problems that involve large amounts of data that require complex functions to deliver the desired application. AEs can learn data nuances if constructed correctly and given enough training samples.

4. Asymmetric autoencoders

The main goal of this paper is to pave the way for the implementation of asymmetrical compression systems based on autoencoders that are more suited for IoT scenarios, i.e., that consider the asymmetrical distribution of computational resources between sensors and cloud or edge-located servers. To accomplish that, we propose new AE architectures where the number of layers or other structures is greater in the decoder than in the encoder. So far, almost all AEs proposed lean on the rule of thumb of constructing AEs where the amount of resources at the encoder and decoder is roughly the same. In these cases, the decoder mirrors the encoder, copying the number and the disposition of layers of the latter. When this is not the case, the additional features tend to be placed in the AE's encoder. Contrary to that, we evaluate if keeping a simple architecture at the encoder while increasing the complexity

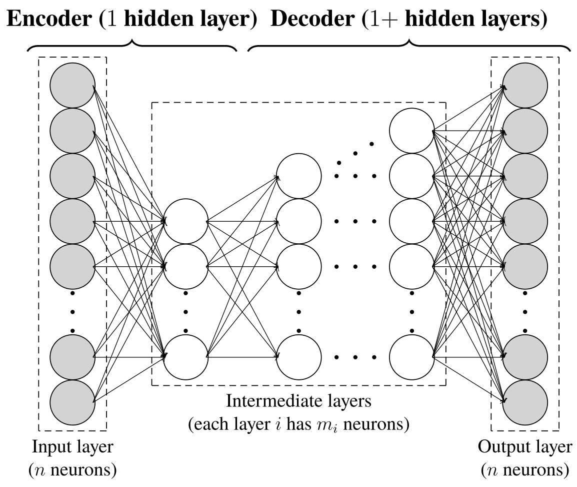


Fig. 2. Proposed Asymmetric AE: unlike typical symmetrical AE, however, the number of decoding layers is greater than the number of encoding layers.

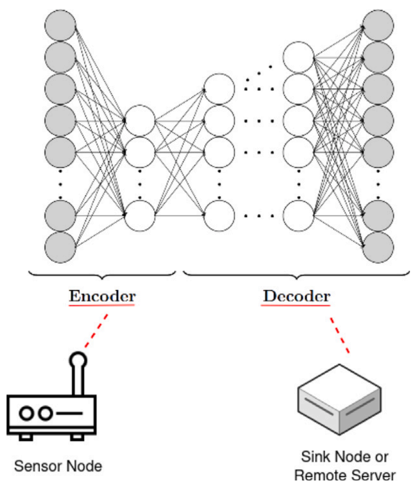


Fig. 3. Layer arrangement of the proposed AAE, where the compact encoder is stored at the sensing node.

of the decoder can bring an equivalent or even better reconstruction rate than in the symmetrical case. If this asymmetrical architecture delivers low reconstruction error, we confirm that additional computational complexity and memory utilization from the sensor perspective is indeed unnecessary.

Fig. 2 illustrates the general picture of the proposed Asymmetric AE (AAE). Even though the model depicts a fully-connected implementation, all AAEs follow the same trend, adding their own tweaks. In the experiments to come, the AAE encoder always has a single fully-connected layer that transforms an array with 100 samples into a compressed representation with 25 values. We have opted to construct all the encoders with this exact strict requirement to facilitate the evaluation of results and to emphasize the contributions of our research. Thus, even though we do not experiment with more complex encoders, we expect that relaxing this requirement leads to even better results. On the other hand, the decoder can have as many layers as we need. Furthermore, as we will discuss, it can incorporate additional structures, such as convolutional layers. Another advantage of constructing all AAEs with a single encoding layer is that it produces an encoder that is a single weight matrix with a bias vector and an activation function. As discussed by Alsheik et al. this configuration offers a lightweight compression scheme, requiring the storage of a few parameters at the sensor node [7]. In this direction, we show that AAEs can retain the advantages for constrained devices and that modifications at the autoencoder's decoder permit performance improvements.

The proposed AAEs are designed for resource-constrained devices acting as sensing nodes in IoT networks. As can be seen in Fig. 3, we place the AAE's encoder at the sensing node, which we assume is a resource-constrained device, and outsource the resource-demanding decoder to a resource-rich device (a remote server, for example). As the results will show, when compared to the symmetrical autoencoders, the proposed AAE is capable of similar or lower reconstruction error than its deeper symmetric counterparts, offering an encoder that can save a significant amount of storage space and computation. All models are trained offline using historical data collected from the phenomena of interest. The machine in charge of training could host the decoder, assuming it has the required computational power, which is a common practice in DNN deployment in IoT and resource-constrained deployments in general [6,17]. Thus, the AAE can be trained in the cloud, if the server has access to the training samples.

AAEs with convolutional layers

We enhanced the AAEs performance by increasing the models' asymmetry with the addition of convolutional layers to the decoder. By adding these layers to the proposed AAEs, it was possible to improve the extraction of intrinsic temporal correlations. Convolutional layers are popular neural network structures that extract similar features across the input data. Compared with the fully-connected layers used earlier, the convolutional layers are composed of a small set of weights that act upon a small neighbouring set of input values. The same strategy is replicated all across the input data [20]. In practice, these weights operate as a filter that analyses small portions of data in a sliding window fashion. In NNs, a kernel usually refers to this set of weights, while the step in which the kernel slides on data is called stride. Additionally, using multiple kernels in a single convolutional layer is a typical approach to extract distinct features in parallel, which can be complementary to one another. To keep the terminology different from the layer's output, one says that the result of each kernel exits an output channel. We adopt these naming conventions throughout this paper.

In the proposed Convolutional AAEs (CAAEs), we employ depthwise-separable-convolutional layers [21]. In these convolutional layers, spatial and cross-channel correlations are extracted separately, meaning that these layers split the extraction of features and mix inter-channel information operations. These layers are shown empirically as capable of learning richer representations with fewer parameters, leaning on the assumption that, by splitting these operations, the whole convolutional layer operation becomes simpler and more efficient [21]. As will be discussed later, a shortage of training samples is a common problem when training NNs with temporal data [22]. Additionally, in early experiments, this seemed particularly beneficial for our denoising models, as we employed a large number of kernels in parallel in our deep CAAE configurations.

Hybrid Dilated Convolution (HDC) [23] is another convolutional layer that we employed in our proposed CAAEs. Using the same input data, the HDC layer extracts features at different scales. Dilated convolution, also referred to as atrous convolution [24], is a variation that employs kernels that skip some adjacent samples to increase the receptive field without loss of resolution or coverage [25]. In this direction, the Hybrid Dilated Convolution (HDC) layer employs different kernels with distinct dilations. For instance, suppose an HDC layer is constructed with kernels $k_1 = (-1, 1, -1)$, $k_2 = (1, 1, 1)$, and $k_3 = (-2, 0, 2)$ with dilations equal to 1, 2 and 3, respectively. In practice, this is similar to a regular convolutional layer with kernels $k'_1 = k_1$, $k'_2 = (1, 0, 1, 0, 1)$ and $k'_3 = (-2, 0, 0, 0, 0, 0, 2)$. Hence, if we apply this HDC to an arbitrary sequence $v = (2, 1, -1, 0, 1, 1, -2, \dots)$, the generated sequences are $(0, -2, 0, \dots)$, $(2, 2, -2, \dots)$ and $(-8, 0, 0, \dots)$, respectively.

Table 2
AEs and AAEs fully-connected decoder configurations.

Label	Layer sizes
AE-0	$25 \rightarrow 100$
(A)AE-1	$25 \rightarrow 50 \rightarrow 100$
(A)AE-2	$25 \rightarrow 50 \rightarrow 75 \rightarrow 100$
(A)AE-3	$25 \rightarrow 45 \rightarrow 65 \rightarrow 85 \rightarrow 100$
(A)AE-4	$25 \rightarrow 40 \rightarrow 55 \rightarrow 70 \rightarrow 85 \rightarrow 100$

5. Experiment setup

We analyse our proposal using temperature readings from a wireless sensor node located at the American River Hydrologic Observatory (ARHO) [26]. The models are expected to capture temporal correlations in data to generate compact representations and eliminate the noise perceived in each sample. In particular, these measurements are from a sensor near Caples Lake in California, taken from June 2014 to October 2017. Consecutive samples are obtained with regular intervals of T = 15 min.

The AE input and output sizes are limited to 100 samples, meaning that the sensor node forwards compressed data corresponding to sample arrays of $s = 100$ temperature readings. Thus, sensor transmissions always occur after the fixed-time interval needed to collect these 100 samples. In the case of the dataset adopted in our experiments, because the samples are collected in intervals of 15 min, one array is composed of more than 1 day of measurements ($T \times s = 1500$ min = 1 day and 1 h). This period of time is not a result of the AEs computation, but rather a limitation of the dataset adopted. Hence, if one wants a faster transmission rate, adopting a shorter period between consecutive samples is sufficient.

5.1. AAE configurations

All networks are designed to reduce the 100 temperature readings collected to a compressed array of size 25. All hidden layers employ SeLU as the activation function, whilst the output layer activation function is the sigmoid function, the latter a common choice for AEs. All fully-connected decoder configurations have a smooth increase in the number of neurons per layer toward the output. We use arrows to highlight the data flow direction for all decoder configurations, as seen in Table 2.

We label our fully-connected autoencoders using the notation “Autoencoder-Model-DepthOfTheVaryingBlock”, adopting the decoder's depth as the identifier for each model. For instance, the first AAE architecture (AAE-1) contains a single hidden decoding layer between the decoder's input (layer with 25 neurons) and output (100 neurons). The symmetric AEs, against which we compare the proposed AAEs, have encoders that mirror the configurations seen in Table 2, as in Fig. 1. In the case of AE-0, it refers to the shallowest symmetrical AE, which has only two NN layers and no extra decoding layer between the compressed and output data. This is the base model, the simplest and starting model for evaluation, which is modified to achieve performance enhancements. Finally, concerning the models' operability, because we adopt sigmoid in the output layer, each sample array is normalized to fall within the sigmoid codomain. Consequently, the maximum and minimum values of the corresponding batch are also transmitted with the compressed data for reconstruction.

5.2. CAAE configurations

The intrinsic temporal correlations, an attribute present in the data used in our experiments, are underutilized in the fully-connected models. Although AAEs exploit them to perform the task at hand, we enhance their use by including convolutional layers. Hence, we can modify the decoder to enhance performance by exploiting the prior knowledge about the sensed signal. These modifications result in the proposed Convolutional AAEs (CAAEs).

All CAAEs proposed in this paper build upon the third decoder configuration (AAE-2) seen in Table 2, with the set of convolutional layers replacing the layer that performs the $75 \rightarrow 100$ transformation. Another feature of the CAAE is that, before the last convolutional block (i.e., set of convolutional layers), the data is upsampled with a factor equal to 2. This means that the convolutional layers employed in our configurations do not promote upscaling. Finally, all data from the last convolutional block is fused together using a convolutional layer with kernels with both size and stride equal to 1. This output layer is the output of the whole network, using sigmoid as the activation function, such as in our previous models.

Table 3 brings the configuration of the convolutional blocks of each CAAE configuration. Similar to Table 2, the first index in the identifier reflects the decoder's depth. Differently, the second index reflects the complexity of the convolutional layers, with the higher identifier reflecting the models with more convolutional kernels. Regarding the layers' characteristics, all kernels have a size equal to 3. Additionally, the convolutional blocks can have an expansion factor of 4 or 8. This value represents the amount of spatial convolutional kernels applied at each input channel. Hence, for n input channels, if the block has t expansion factor, the total number of kernels is nt. Similarly to the hidden layers of the previous AAEs, we employ SeLU as the activation function.

5.3. Dataset generation

All networks are trained following the same methodology. Samples until June 2016 form the training set, whilst the testing set contains the remaining samples. This meant that approximately 60% of samples were used for training, and the remaining 40% for testing. After splitting the samples into training and testing sets, arrays of 100 samples are generated in a sliding window fashion. The goal is to create a significantly large number of distinct training arrays in comparison to a simple training set partition, given that the lack of training data is a common problem with temporal data [22]. In particular, three distinct strides are selected to increase the number of training arrays together. We, however, do not conduct any particular analysis to determine the optimal stride, using 10, 17, and 23 instead, given that they are mutually prime. Any duplication is removed to ensure training array uniqueness.

We analyse two main scenarios, one where the models are trained to perform just data compression, and another where the models must also perform data denoising. In the latter, the training data is corrupted with Additive White Gaussian noise (AWGN) to enable the models to learn how to remove the noise. Specifically, each array in the training dataset can be contaminated with noise at levels ranging from 2.5 to 80 db SNR. After generating the training set, all arrays are shuffled.

5.4. Training specifications

All models are trained using the early stopping policy, meaning that the training process terminates after a given number of epochs without improvements. We set the patience to be equal to 15 epochs. Additionally, we employ Nadam [27] as our optimizer, which is a variation of Adam [28] optimizer, a popular optimizer for NN training employing Nesterov's momentum [29]. We adopt decrease on the plateau as the learning rate (LR) scheduling policy, meaning that after a number of consecutive epochs without improvement (we set this value to be equal to 5) the LR multiplied by a factor of 0.75. This process is repeated until reaching a minimum LR. In our experiments, the starting and ending LR's are equal to $3 \cdot 10^{-3}$ and $5 \cdot 10^{-5}$, respectively. During training, the input of the models were batches of training data formed by 50 arrays.

Unlike most works that employ autoencoders for data denoising [6, 12], we did not include any regularizer and relied on the network's

Table 3
CAAE configurations.

Label	1st Conv. Block			2nd Conv. Block			3rd Conv. Block		
	nout	exp	dil	nout	exp	dil	nout	exp	dil
CAAE-3.1	4	4	1	4	4	1	–	–	–
CAAE-3.2	8	4	1	8	8	1	–	–	–
CAAE-4.1	4	4	1	[4, 2, 2]	[4, 4, 4]	[1,2,3]	8	4	1
CAAE-4.2	8	4	1	[4, 2, 2]	[8, 8, 8]	[1,2,3]	8	8	1

architecture instead. The models learn the adequate compressed representation and how to recover the noiseless signal by measuring the difference between its output and the uncorrupted original signal. We considered the addition of a regularizer unnecessary for the presentation of our results. Nonetheless, we expect that the adoption of such regularizations does not negatively impact the AAE training.

The code used to derive the experimental results is written in Python, built using the TensorFlow framework [30]. It is made available at a GitHub repository [31]. Each model is trained 10 times, using different initial weights. The performance of each model is evaluated by computing the average reconstruction error over all testing arrays. The reconstruction error of each array of samples is measured using the Mean Square Error (MSE). Finally, in the case of the models trained to perform noise suppression, we assess their robustness to noise by adding AWGN to the test dataset.

6. Experimental results

The experiments in both scenarios (compression with and without denoising) are divided into three. The first evaluates the symmetric AEs, analysing how increasing the model's depth affects reconstruction error. Also, we show the space and computational costs of resorting to the symmetrical approach, as both the number of encoding parameters and floating-point operations (FLOPs) increase when we add more layers to the AE's encoder. The second compares the performance of AEs and AAEs, comparing the proposed AAEs with the symmetrical ones we want to replace. We show the advantages of seeking NN-depth increase without incurring more encoding layers, as they have similar performance while requiring fewer parameters and FLOPs. Finally, the third experiment evaluates how one can exploit data temporal correlation using CAAEs to achieve additional performance gains. These models show that, in addition to increasing decoder depth, we can pursue performance enhancements by adding features that were absent in the encoder. Moreover, by varying the number of kernels per convolutional layer, we can reduce the reconstruction error by implementing other modifications without solely resorting to additional NN layers. Finally, specifically to the results shown here, the testing arrays are constructed using strides equal to 33 and 50, which are respectively roughly a third and half the size of the arrays of samples. As in the case of the training arrays, duplicated testing arrays are discarded.

6.1. Data compression

6.1.1. Increasing AE depth

Fig. 4 brings a boxplot showing the distribution and the mean reconstruction error for all analysed symmetric AEs. It clearly shows the advantages of increasing the number of layers, especially compared to AE-0. Apart from a single model that delivered a mean reconstruction error close to 0.176 MSE, all AE-0 models were unable to outperform any of the deeper AEs. Moreover, the models that delivered the lowest reconstruction error are those with more NN layers (AE-2 and deeper configurations). However, when observing the median (red horizontal line segment) and the mean (dark red diamonds) of the models with two or more hidden decoding layers, we notice that there is no significant gain when opting for a model with more than two hidden decoding (and encoding) layers. Notice that, by increasing the number of layers symmetrically, we are adding two more layers compared to the previous configuration, given that we are adding one layer at the encoder and another at the decoder. Hence, the deeper models have a significant amount of parameters to adjust, requiring more data to train. This is another problem that may occur when deploying deep-symmetrical-AE architectures in IoT networks, as the amount of data on a particular phenomenon can be limited. As will be seen later with our proposed models, this issue is minimized significantly.

Table 4
Number of parameters and FLOPs needed for each compression in each AE encoders.

Conf.	Numb. Params.	FLOPs
AE-0	2525	5025
AE-1	6325	12 575
AE-2	12 650	25 150
AE-3	18 295	36 370
AE-4	21 775	43 275

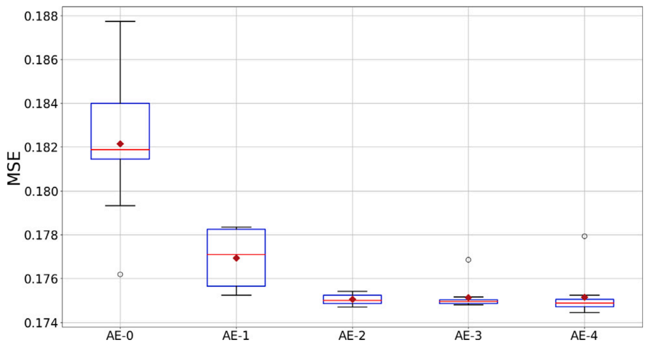


Fig. 4. Mean reconstruction error for all symmetric AEs.

Before moving on to the AAEs, it is important to quantify the increase in size of the encoders. Table 4 brings the number of trainable parameters (weights and biases) in the encoder of each AE configuration. The number of parameters grows considerably as the number of layers increases. For instance, to achieve the performance gains in Fig. 4, the number of encoding parameters more than doubles from AE-0 to AE-1, and doubles going from AE-1 to AE-2. Moreover, this issue tends to worsen when dealing with more complex data, which requires more layers and neurons. The proposed AAEs are an alternative to minimize these problems. Even if quantization is adopted later, starting with a model that is already more compact can lead to even more space conservation. For instance, this can be beneficial when envisioning that multiple models can be stored in a sensing node to deal with data collected from multiple phenomena [3].

Another issue with increasing encoder size is the increase in encoding computations. Table 4 brings an approximation of FLOPs for each encoder (ignoring the FLOPs that SeLU may impose), which gives a good estimate of the number of computations needed to generate one compressed array. For instance, when opting for AE-1, the amount of FLOPs more than doubles in comparison with AE-0. Moreover, if we opt for AE-2, the resulting encoder imposes FLOPs that are more than five times higher than those of AE-0. Adopting a similar analysis as Alsheikh et al. [7], which computed the power consumption in an MSP430 microcontroller, these increase in FLOPs leads to about

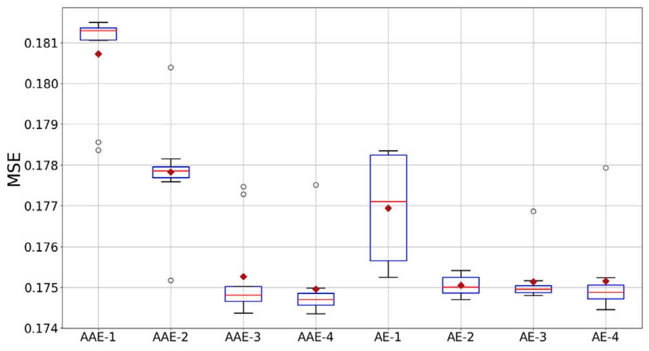


Fig. 5. Comparison between the mean reconstruction error distribution of the proposed AAEs against their symmetrical counterparts.

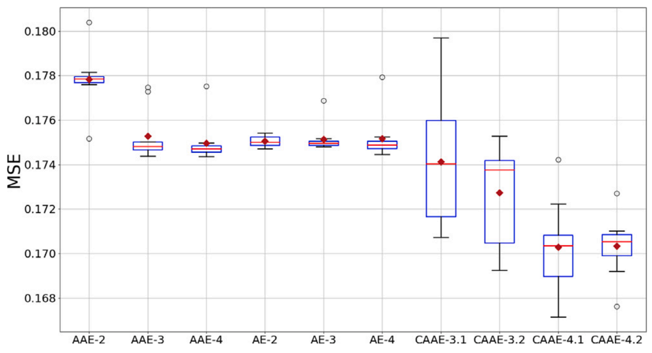


Fig. 6. The mean reconstruction error distribution of the proposed CAAEs compared with the best-performing AEs and AAEs.

4.02 mJ and 10.71 mJ more energy spent in each data compression Appendix. AE-0 spends 2.65 mJ per compression. Thus, opting for AE-1 nearly triples power consumption, and AE-2 spends more than five times more energy. This trend tends to worsen with more complex data because they will likely require more complex models. Additionally, other factors, such as sampling rate, can increase sensor node power consumption. We address this computational problem by showing that performance improvements are possible even when we keep AE-0's encoder configuration, thus showing that these implementation issues can be mitigated through a different autoencoder design.

6.1.2. *AEs vs. AAEs*

Fig. 5 shows that the AAEs keep the trend of improvement as we increase the number of layers. Recall that the AAEs have the same encoder configuration as AE-0, meaning that they have the same number of encoding parameters and impose the same number of FLOPs shown in Table 4. The additional layers are inserted in the decoder, to be implemented away from the resource-constrained devices. However, it is noticeable that the advantage of the asymmetric design starts to appear with AAE-2, when two layers are added to the decoder. Interestingly, this is the AAE with the same number of NN layers as AE-1 (both have 5 NN layers in total), and they have comparable performance. For example, it is noticeable that their medians and means are less than 0.001 MSE apart. Considering computation demands, AAE-2 is more advantageous for resource-constrained implementations because it requires 5025 FLOPs per data compression, whilst AE-1 takes 12575.

Moving on to the deeper architectures, the advantages of the proposed AAEs become more evident. AAE-3 and AAE-4 not only have comparable performance to AE-2 and deeper configurations but are also capable of outperforming them while requiring five times fewer parameters and FLOPs, at least. First, they are capable of delivering a similar performance, evidenced by the observation that AAE-4 and AE-4 have similar error distributions, with AAE-4 having a slightly better performance. For instance, the third quartile of AAE-4 (topmost side of the box) is almost equal to the AE-4 median. Additionally, we can observe that some deeper AAEs are capable of outperforming most of all trained AEs. Observer in the graph that the median of both AAE-3 and AAE-4 are below the medians of AE-3 and AE-4. Given that the AAEs have an encoder with the same size as AE-0, they appear a better choice for the analysed scenario. Opting for either of the two AAE configurations instead of AE-3 and AE-4 means that we are saving seven times fewer FLOPs per compression operation, at least. Thus, AAEs can be a valuable alternative for resource-constrained devices.

6.1.3. *Exploiting temporal correlation with CAAEs*

Fig. 6 shows the performance of the CAAEs, comparing them with the best models so far. These results show the advantages of adding convolutional layers to the asymmetrical architecture. To illustrate this, notice that the models achieving the best results among all configurations are constructed with HDC blocks, and that even the worst-performing CAAE models outperform all AEs and AAEs seen previously. This is another positive result for the proposed asymmetrical architecture, emphasizing that we can seek performance improvements by adding new features to the decoder that were not present in the encoder. However, Fig. 6 shows that the results obtained with CAAEs have more variance than those obtained with our previous architectures. For the sake of fair comparison, we adopted the same training parameters to train all models. Hence, it is possible to minimize this variance by adopting more appropriate training parameters to exclusively train CAAEs.

Analysing solely the CAAEs results, the fact that the models with an HDC block present the best results highlights the advantages of employing a layer with distinct dilations. By doing so, we ensure that the kernels learn to extract distinct features, because the kernels are operationally distinct. Moreover, these features are complementary, given they are obtained in different resolutions. We know that temporal data have short- and medium-term relations, meaning that a CAAE with HDC is more suited to extract them than a CAAE without this convolutional block.

When comparing CAAE-4.1 and CAAE-4.2, we observe that adding more layers (through the expansion factor) seems unnecessary. This appears similar to the results in Fig. 5, when comparing the deeper AEs and AAEs configurations. However, as will be seen in our next experiments with noise, this can be attributed to the problem's relative simplicity rather than a shortage of training data. For example, increasing the number of kernels is advantageous when comparing the performances of CAAE-3.1 and CAAE-3.2, which have convolutional kernels with the same dilation. Compression without denoising is a more straightforward task, suggesting that having a lot of convolutional kernels becomes unnecessary when deploying the HDC block, given its improved capacity to extract distinct and complementary features.

6.1.4. *Reconstruction error and encoder requirements*

In addition to our previous results, Fig. 7 shows the mean reconstruction error of each autoencoder configuration considering the number of FLOPs needed for each data compression. In it, we can see the advantages of opting for the asymmetrical approach. First, we observe that adding more layers and features to the AAE's decoders enhances the model's performance without incurring more FLOPs. Additionally, as the CAAEs results show, we can add new features to the decoder to improve data reconstruction, allowing us to outperform the symmetric and costly AEs. Overall, combining these results with the ones previously discussed, we can see that AAEs can offer designers a more compact model capable of rivalling the deep autoencoder configurations without increasing the encoder implementation costs, which can help its implementations in resource-constrained devices.

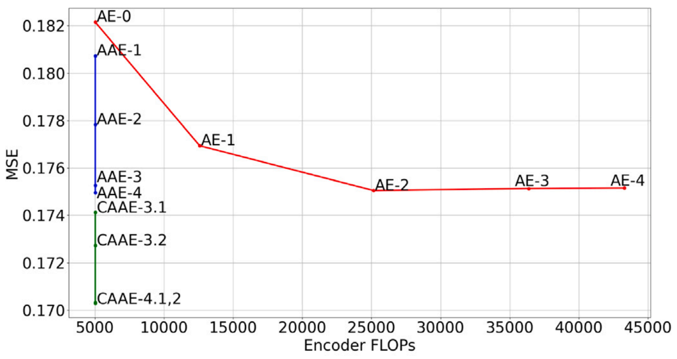


Fig. 7. Comparison between proposed AAEs and AEs reconstruction error and encoder FLOPs demands. Performance improvements are possible without increasing the encoder's FLOPs, with CAAEs outperforming the deep AEs.

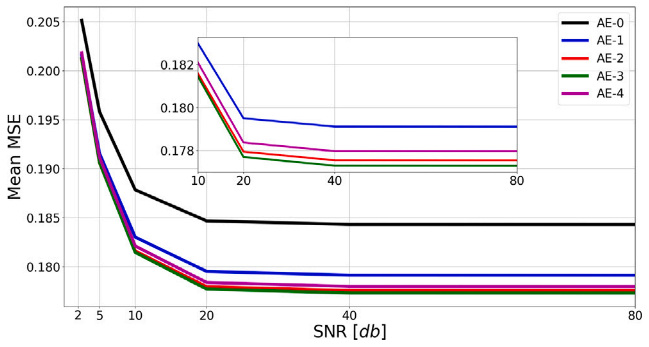


Fig. 8. Mean reconstruction error for all symmetric AEs.

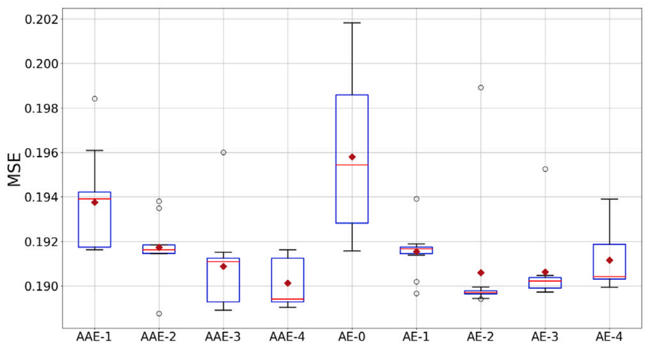
6.2. Data compression with denoising

Requiring models to also learn how to perform data denoising presents a more challenging training scenario. In addition to learning how to compress data, all models need to learn how to remove noise from that data. Our experiments show that this is particularly challenging for the models with multiple trainable parameters, which need a lot of data to adjust their parameters. Hence, in this scenario, data shortage problems, previously seen as an issue when training models with multiple layers, tend to be aggravated.

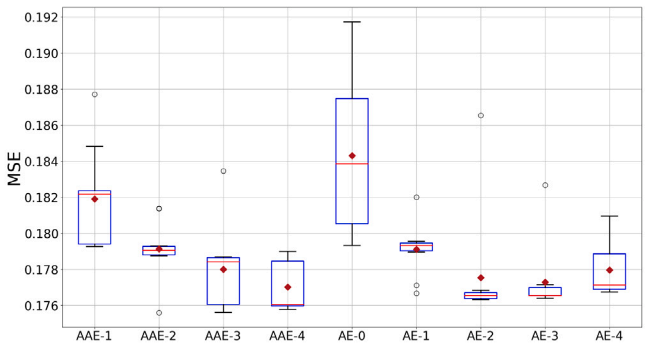
6.2.1. Increasing AE depth

Fig. 8 shows the mean reconstruction error for all symmetric AEs. The first noticeable result in this figure consists of all deep architectures outperforming AE-0 across all SNRs by a considerable margin. For example, even when AE-0 is subject to high SNR levels, it offers a higher reconstruction error than the AEs deeper than it in worse noise conditions. All AEs except AE-0 deliver a mean error below 0.185 MSE under SNR levels equal to 10 db, whereas the base model's mean error at 80 db is almost equal to this value. Later, when comparing AEs with AAEs, it becomes clear that our proposed models are fundamental to enabling autoencoders with a single encoding layer to match or even surpass the performance of those with multiple encoding layers.

The second noticeable result is that AE-2 and AE-3 outperform AE-4, which is the deepest AE configuration. The difference is even more significant when comparing AE-3 with AE-4. A more in-depth analysis of the AEs performance under SNR equal to 5 db and 40 db, seen in Fig. 12, shows that most of the models constructed with the AE-2 architecture deliver better reconstruction errors than the deeper symmetrical models. This is particularly pronounced in the scenario with more noise, as most AE-2 models are among the best-performing symmetric models (Fig. 9(a)). For instance, under 5 db SNR, nearly



(a) Reconstruction error with SNR=5 db.



(b) Reconstruction error with SNR=40 db.

Fig. 9. Reconstruction error for all fully-connected autoencoders.

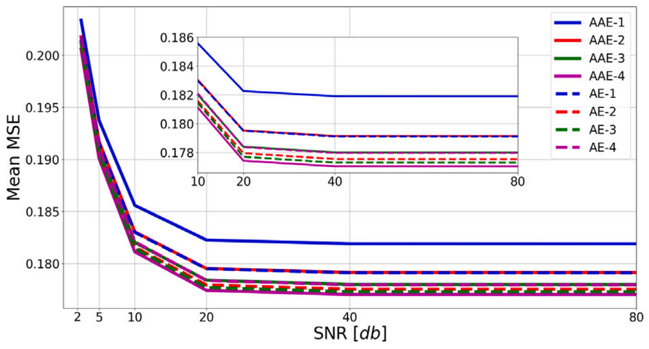


Fig. 10. AAEs mean reconstruction error results compared with the symmetric counterparts. Models with the same colour scheme have the same decoder layout.

all AE-2 models deliver a lower reconstruction error than all AE-4 models, and surpass at least half of the AE-3 models, given that they are below AE-3's median. This shows another limitation of employing deep NN architectures to deal with problems with low availability of training data. As highlighted earlier, the scenario at hand here is more challenging, requiring that the models also learn how to remove noise from the sensed signal. This suggests that more samples are needed to train deeper models to perform both tasks simultaneously. Again, we highlight that the necessity of more training samples is another issue that can be worked around with our proposed models, as they have fewer adjustable parameters than the symmetric models.

6.2.2. AEs vs. AAEs

Fig. 10 shows the mean reconstruction error for the proposed AAEs (continuous lines), comparing them with their symmetric counterparts (dashed lines). Focusing solely on the models with 2 hidden decoding layers or less, we can observe that shallow AAEs do not fall behind their symmetric counterparts. Furthermore, AAE-2 was able to deliver

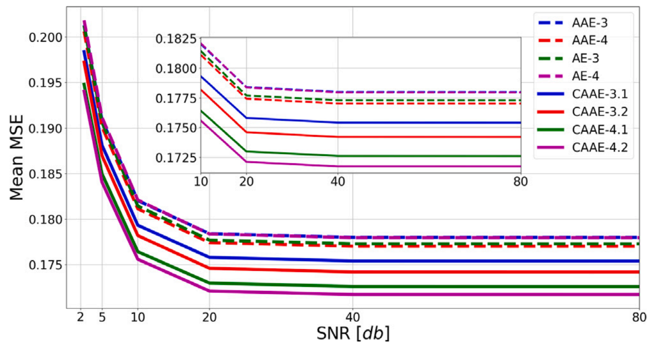


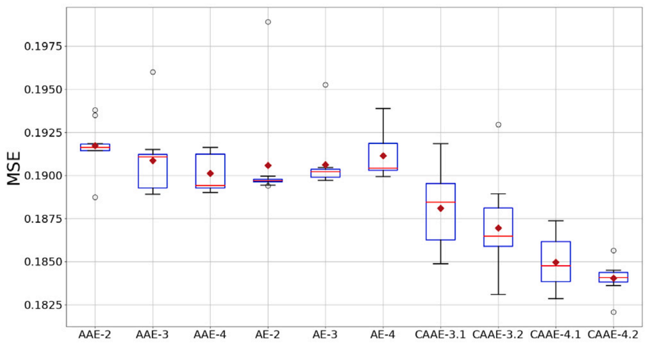
Fig. 11. CAAEs mean results, in comparison with the AAEs and AEs with similar decoder depth.

a performance nearly identical to AE-1. Additionally, unlike observed with AEs, the deeper models keep the improvement trend as the number of layers increases, observable in the noiseless scenario. The error obtained with a deep AAE is always below shallower configurations across all SNR levels above 5 db, which was not the case with the AEs. This is more evident when we observe the boxplots in Fig. 9. Moreover, noticing that the median of AAE-4 in both plots is below all configurations, we observe that the deepest AAE architecture outperforms all AEs models. These results suggest that the training issues discussed in the previous subsection are minimized by the proposed AAEs. These findings show that AAEs fit also well to IoT compression scenarios that need noise suppression. Another result shown in Fig. 10 is that for SNR ranging from 20 db to 80 db, the performance of all models is seemingly the same. Also, the performance drop from 20 db to 10 db is not very eminent, in comparison with the drop from 10 db to 5 db and from the latter to 2.5 db. Both results indicate the success of the proposed training methodology. However, we expect that the adoption of regularizers to assist the AAEs in learning denoising can improve the results for higher noise levels.

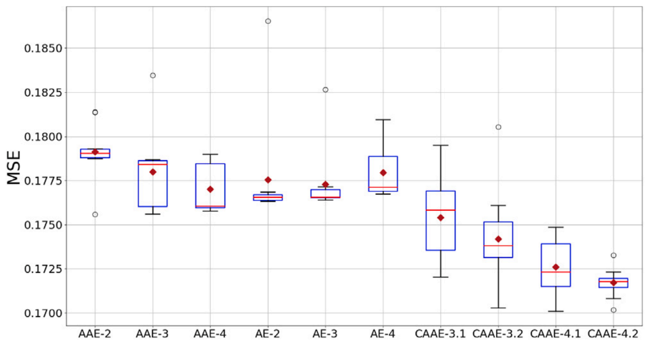
Fig. 9 shows that the relative performance of all models has minimal changes when the noise level increases. The overall performance of all AEs and AAEs is similar in both graphs, and there is also consistency in the variance among autoencoders that have the same architecture. This suggests that models that perform well at low noise levels tend to deliver the smallest reconstruction error in worse noise conditions. However, as discussed earlier when AE-2 performance at $5db$ was compared with the deeper AE models, the loss in performance is more pronounced with the deeper AE architectures. This is not an issue with the AAEs, as the deeper models that performed well at high SNR levels still outperform the other AAEs when the SNR levels are lower. This suggests that the deeper AAEs are less sensitive to noise than their symmetrical counterparts. Thus, one can expect that the AAEs are well-suited to concurrently perform the compression and denoising tasks simultaneously. Besides having an encoder with fewer parameters to store and computations to perform, the fact that they have fewer layers overall in comparison with the symmetric ones means that they require fewer training samples to adequately learn complex tasks.

6.2.3. Exploiting temporal correlation with CAAEs

Fig. 11 shows the mean reconstruction error of the proposed CAAEs in comparison with the AEs and AAEs with similar decoder depth, *i.e.*, models with three or more hidden decoding layers. Note that the CAAEs continue the trend of performance improvement of the proposed asymmetrical architectures. Furthermore, combining these results with those found with the other AAE models, and noticing that all the proposed asymmetrical architectures have the same encoder configuration as AE-0, we call attention to the fact that the main contribution of our proposed asymmetric architecture is opening new opportunities for performance improvement without relying on more encoding layers.



(a) Reconstruction error with SNR=5 db.



(b) Reconstruction error with SNR=40 db.

Fig. 12. In-depth look of best-performing autoencoders.

So, by constructing decoders with enough resources and capable of exploiting the distinctive features of the phenomenon of interest, one can design autoencoders that can overcome the lack of encoding layers.

Fig. 12 presents a detailed analysis of the CAAEs alongside the best-performing autoencoders seen previously. Similarly to Fig. 9, the overall dispersion of the CAAEs under different noise levels is almost unchanged. This is another result that backs our assumptions that asymmetric autoencoders performing well with low noise levels tend to be the best models when the level of noise increases. However, Fig. 12 brings an additional result when compared with Fig. 6. Different from the latter, where adding more kernels seemed irrelevant, CAAE-4.2 benefits from the greater number of filters to extract the needed features to succeed in the denoising task. This suggests that one can experiment with adding more kernels to the convolutional blocks, in order to assess if more filters can improve the CAAEs performance in a more complex task. Given that these changes are done to the decoder, the resource-constrained device where the encoder will be implemented is oblivious to changes in the CAAEs architecture.

7. Conclusion

An important concern when implementing deep-learning-based solutions in IoT networks is the burden they may pose to the sensing nodes. Typically, neural network (NN) models require significant storing and computational resources that resource-constrained devices may not provide. Therefore, there is a need to explore alternative NN architectures. Recently, many solutions that resort to autoencoders are being proposed to handle IoT data. Among them, autoencoders were shown to be useful in data compression and noise suppression applications. However, the typical autoencoder (AE) architecture does not scale well for IoT deployment, as increasing the number of layers in the encoder poses a problem for implementing them in sensor nodes.

Hence, to alleviate NN's implementation in this scenario, we propose the Asymmetric Autoencoder (AAE), an autoencoder variation in which the number of layers (and other computational resources) is greater in the decoding block of the model than in the encoding block, offering an alternative that can shift the bulk of the computation away from sensor nodes.

The results show that the adoption of the proposed AAEs suits well IoT sensing. These asymmetric models have encoders with fewer parameters and that require fewer FLOPs per data compression operation. The AAEs are capable of delivering reconstruction errors that can rival their symmetric counterparts, and can even outperform them. Additionally, the AAEs seem to be more suited for the scarcity of training samples, a problem that may appear in many IoT settings. This assertion is based on results that show that the deeper AAEs keep the performance improvement with depth increase in both scenarios analysed, whereas symmetric AEs fail to keep this improvement trend. Especially when trained to perform denoising together with compression, the deeper AE models stopped showing significant improvements, likely due to the shortage of training samples. Furthermore, the results of the Convolutional AAEs (CAAEs) show that we can pursue further performance improvements by modifying the decoders to exploit the particularities of the phenomenon of interest. Given that the signal we used in our experiments has temporal correlations, we minimized the reconstruction error of the recovered signal by adding convolutional layers to the decoder. Hence, the main result presented in this paper is that one can improve feature extraction, overcoming the lack of encoding layers, by adding more resources to the decoding block and exploiting prior knowledge about the sensed signal. This offers an alternative approach that can be helpful when designing AE-based solutions for resource-constrained devices in IoT networks.

CRediT authorship contribution statement

Mateus S. Gilbert: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Marcello L.R. de Campos:** Funding acquisition, Project administration, Resources, Supervision, Visualization, Writing – original draft, Writing – review & editing, Conceptualization. **Miguel Elias M. Campista:** Conceptualization, Funding acquisition, Project administration, Resources, Supervision, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It was also supported by CNPq, Brazil, FAPERJ, Brazil Grants E-26/211.144/2019 and E-26/202.689/2018, and FAPESP, Brazil Grant 15/24494-8. Additional support granted by Development and Research Foundation - Fundep - Rota 2030.

Appendix. Energy consumption estimation

According to Alsheikh et al. [7], one clock cycle in an MSP430 microcontroller accounts for 1.85 nJ. Additionally, each multiplication and addition operation requires 395 and 184 clock cycles, respectively. A matrix multiplication between an $M \times N$ matrix and an $N \times 1$ vector takes $M \times N$ multiplications and $M \times N - 1$ additions. Hence, in each encoding layer, we have

$$(395MN + 184M(N-1) + 184N) \cdot 1.85 \text{ nJ}.$$

Thus, recalling the encoder configurations in Table 2,

- AE-0 consumes approximately 2.65 mJ;
- AE-1 consumes approximately 6.67 mJ;
- AE-2 consumes approximately 13.36 mJ.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.

[2] S. Tilak, N.B. Abu-Ghazaleh, W. Heinzelman, A taxonomy of wireless micro-sensor network models, ACM SIGMOBILE Mob. Comput. Commun. Rev. 6 (2) (2002) 28–36.

[3] J. Azar, A. Makhoul, M. Barhamgi, R. Couturier, An energy efficient iot data compression approach for edge machine learning, Future Gener. Comput. Syst. 96 (2019) 168–175.

[4] J.D.A. Correa, A.S.R. Pinto, C. Montez, Lossy data compression for iot sensors: A review, Int. Things 19 (2022) 100516.

[5] M.F. Duarte, G. Shen, A. Ortega, R.G. Baraniuk, Signal compression in wireless sensor networks, 2012, http://dx.doi.org/10.1098/rsta.2011.0247.

[6] K. Bochie, M.S. Gilbert, L. Gantert, M.S. Barbosa, D.S. Medeiros, M.E.M. Campista, A survey on deep learning for challenged networks: Applications and trends, J. Netw. Comput. Appl. (2021) 103213.

[7] M.A. Alsheikh, S. Lin, D. Niyato, H.-P. Tan, Rate–distortion balanced data compression for wireless sensor networks, IEEE Sens. J. 16 (12) (2016) 5072–5083.

[8] T. Yu, X. Wang, A. Shami, Uav-enabled spatial data sampling in large-scale IoT systems using denoising autoencoder neural network, IEEE Internet Things J. 6 (2) (2018) 1856–1865.

[9] G. Li, S. Peng, C. Wang, J. Niu, Y. Yuan, An energy-efficient data collection scheme using denoising autoencoder in wireless sensor networks, Tsinghua Sci. Technol. 24 (1) (2018) 86–96.

[10] A.M. Ghosh, K. Grolinger, Deep learning: Edge-cloud data analytics for IoT, in: 2019 IEEE Canadian Conference of Electrical and Computer Engineering, CCECE, IEEE, 2019, pp. 1–7.

[11] M.A. Razzaque, C. Bleakley, S. Dobson, Compression in wireless sensor networks: A survey and comparative evaluation, ACM Trans. Sensor Netw. 10 (1) (2013) 1–44.

[12] F. Laakom, J. Raitoharju, A. Iosifidis, M. Gabbouj, Reducing redundancy in the bottleneck representation of the autoencoders, 2022, arXiv preprint arXiv:2202.04629.

[13] W.-H. Lee, M. Ozger, U. Challita, K.W. Sung, Noise learning-based denoising autoencoder, IEEE Commun. Lett. 25 (9) (2021) 2983–2987.

[14] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for iot big data and streaming analytics: A survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960.

[15] A. Majumdar, A. Tripathi, Deep learning: Edge-cloud data analytics for IoT, in: 2019 IEEE Canadian Conference of Electrical and Computer Engineering, CCECE, IEEE, 2019, pp. 1–7.

[16] H. Han, J. Siebert, Tinyml: A systematic review and synthesis of existing research, in: 2022 International Conference on Artificial Intelligence in Information and Communication, ICAIIC, IEEE, 2022, pp. 269–274.

[17] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, A.S. Hafid, A comprehensive survey on tinyml, IEEE Access (2023).

[18] D. Charte, F. Charte, S. García, M.J. del Jesus, F. Herrera, A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines, Inf. Fusion 44 (2018) 78–96.

[19] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[20] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, in: The Handbook of Brain Theory and Neural Networks, Vol. 3361, (10) 1995, p. 1995.

[21] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1251–1258.

[22] B.K. Iwana, S. Uchida, An empirical survey of data augmentation for time series classification with neural networks, Plos One 16 (7) (2021) e0254841.

[23] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, G. Cottrell, Understanding convolution for semantic segmentation, in: 2018 IEEE Winter Conference on Applications of Computer Vision, WACV, Ieee, 2018, pp. 1451–1460.

[24] L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation, 2017, arXiv:1706.05587.

[25] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, 2015, arXiv preprint arXiv:1511.07122.

[26] R. Bales, G. Cui, R. Rice, X. Meng, Z. Zhang, P. Hartsough, S. Glaser, M. Conklin, Snow depth, air temperature, humidity, soil moisture and temperature, and solar radiation data from the basin-scale wireless-sensor network in American river hydrologic observatory (ARHO), 2020, http://dx.doi.org/10.6071/M39Q2V, (Accessed in 08/06/2020).

[27] T. Dozat, Incorporating nesterov momentum into adam, in: International Conference on Learning Representations, ICLR, 2016.

[28] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[29] Y.E. Nesterov, A method for solving the convex programming problem with convergence rate o (1/kˆ 2), in: Dokl. Akad. Nauk Sssr, Vol. 269, 1983, pp. 543–547.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, [TensorFlow]: Large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org. https://www.tensorflow.org/.

[31] M.S. Gilbert, Neural network temporal compression, 2023, https://github.com/MateusGilbert/nn_temp_compression, (Accessed in 07/21/2023).

Mateus S. Gilbert is a M.Sc. student in Electrical Engineering at the Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia of the Universidade Federal do Rio de Janeiro (COPPE/UFRJ), Rio de Janeiro, Brazil, since 2021. In 2022, he received his B.Sc. in Electronic and Computer Engineering at the UFRJ Polytechnic School and B.Math at the UFRJ Intitute of Mathematics. His research interests are neural network and deep learning, IoT, mobile networks, computer vision, cloud and fog computing, and data science.

Marcello L.R. de Campos received the Engineering degree (cum laude) from the Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, in 1990, the M.Sc. degree from COPPE/UFRJ in 1991, and the Ph.D.degree from the University of Victoria, Victoria, BC, Canada, in 1995, all in electrical engineering. He is currently Vice-Director of COPPE/UFRJ and Professor in the Program of Electrical Engineering, where he has also served as Department Vice-Chair and Chair in 2004 and 2005, respectively. His research interests include adaptive signal processing in general and its application to distributed networks in particular, adaptive beamforming, statistical signal processing, statistical and machine learning, optimization, signal processing for communications, underwater, mobile and wireless communications, and MIMO systems.

Miguel Elias M. Campista is an associate professor at the Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, since 2010. His major research interests include network and data science, wireless networking, and cloud and fog computing. Campista received his D.Sc. degree in Electrical Engineering from UFRJ in 2008 and spent 2012 in the LIP6 lab at Sorbonne Université, Paris, France, as invited professor. He is a former affiliate member of the Brazilian Academy of Sciences and an associate editor of Annals of Telecommunications, Springer.