Predicting Hard Disk Failures in Data Centers Using Temporal Convolutional Neural Networks

Predicting Hard Disk Failures in Data Centers using Temporal Convolutional Neural Networks

Alessio Burrello[1]✉, Daniele Jahier Pagliari[2], Andrea Bartolini[1], Luca Benini[1,4], Enrico Macii[3], and Massimo Poncino[2]

[1] Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy
{name.surname}@unibo.it

[2] Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy
{name.surname}@polito.it

[3] Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino, Turin, Italy
{name.surname}@polito.it

[4] Department of Information Technology and Electrical Engineering at the ETH Zurich, 8092 Zurich, Switzerland
lbenini@iis.ee.ethz.ch

Abstract. In modern data centers, storage system failures are major contributors to downtimes and maintenance costs. Predicting these failures by collecting measurements from disks and analyzing them with machine learning techniques can effectively reduce their impact, enabling timely maintenance. While there is a vast literature on this subject, most approaches attempt to predict hard disk failures using either classic machine learning solutions, such as Random Forests (RFs) or deep Recurrent Neural Networks (RNNs).

In this work, we address hard disk failure prediction using Temporal Convolutional Networks (TCNs), a novel type of deep neural network for time series analysis. Using a real-world dataset, we show that TCNs outperform both RFs and RNNs. Specifically, we can improve the Fault Detection Rate (FDR) of $\approx 7.5\%$ (FDR $= 89.1\%$) compared to the state-of-the-art, while simultaneously reducing the False Alarm Rate (FAR $= 0.052\%$). Moreover, we explore the network architecture design space showing that TCNs are consistently superior to RNNs for a given model size and complexity and that even relatively small TCNs can reach satisfactory performance. All the codes to reproduce the results presented in this paper are available at https://github.com/ABurrello/tcn-hard-disk-failure-prediction.

Keywords: Predictive maintenance, IoT, Deep Learning, Sequence analysis, Temporal Convolutional Networks

1 Introduction

The storage systems of modern data centers can easily include from thousands to millions of hard disk drives. Therefore, despite single hard drives having a very high Mean Time To Failure (MTTF), storage system failures remain one of the main contributors to data center downtimes and maintenance costs [17, 15]. One solution to reduce the impact of such failures is to resort to predictive maintenance techniques, where disks operations are continuously monitored. An alarm is raised whenever a drive is predicted to fail shortly, hence allowing timely maintenance actions (replacement of the hard drive, data integrity restoration, etc.)[3, 18, 19, 14, 2, 1].

Most hard disk producers adopt Self-Monitoring Analysis and Reporting Technology (SMART) as the tool to enable predictive maintenance [4]. SMART consists of the periodic collection of a set of measurements (SMART *features*) during disk operation, including temperature, seek and read errors, reallocated sectors, etc. These features are then analyzed to determine the likelihood of failure, typically with data-driven approaches based on machine/deep learning models. In particular, most state-of-the-art methods are either based on Random Forests (RFs) or Recurrent Neural Networks (RNNs) [3, 18, 19, 14, 2, 1].

Recently, however, many works [12, 5] have demonstrated the superiority of Temporal Convolutional Networks (TCNs) for time series analysis. TCNs are particular types of 1-dimensional Convolutional Neural Networks (CNNs), including specific architectural elements (causality and time-dilation) to better adapt to time series. In [12, 5], TCNs have been shown to outperform the more expensive and complicated RNNs in many sequence modeling tasks.

In this paper, we assess the effectiveness of TCNs for predicting hard disks failures. To the best of our knowledge, this is the first work to consider Temporal Convolutional Networks for this task. The contribution of this work is three-fold: *i)* we describe a comprehensive analysis of the imbalance management of the failure prediction in hard drives, demonstrating that using a Synthetic Minority Over-sampler improves the performance of up to 43.5% for different classification algorithms.*ii)* We show that the proposed TCN can outperform both RFs and RNNs for failure prediction, mostly thanks to the excellent long-term memory of these networks, which allows them to benefit from a long history of input data. Specifically, with a 90-days input window, we can improve the Fault Detection Rate (FDR) by ≈7.5% compared to state-of-the-art methods, while simultaneously reducing also the False Alarm Rate (FAR) to 0.052%. *iii)* We explore the architectural design space to provide a family of models that offer different trade-offs in terms of processing complexity and memory occupation versus performance. In doing so, we also show that TNCs are consistently superior to RNNs for a given size or complexity. The codes used in all the analysis are public at https://github.com/ABurrello/tcn-hard-disk-failure-prediction.

2 Background and Related Works

2.1 Temporal Convolutional Networks

The main layer type included in TCNs is a particular 1-D Convolution, with two differences compared to standard CNNs:
(1) *Causality*, which implies that each layer output is produced looking only at past samples, i.e. y_{t_N} is obtained from the convolution of x_{t_I} with $t_I \leq t_N$.
(2) *Dilation*, which is the fundamental element behind the success of these nets. Rather than performing convolution on a contiguous time-window of input samples, dilation inserts a fixed step between convolution inputs, thus increasing the receptive field, while keeping the number of parameters low.
Formally, a 1-D causal dilated convolution is computed as:

$$\mathbf{y}_T^m = \text{Dilated-Conv}\left(\mathbf{x}\right) = \sum_{n=0}^{N-1} \sum_{t=0}^{K-1} \mathbf{x}_{T-d\,t}^n \cdot \mathbf{W}_t^{n,m}$$

where $\mathbf{x}$ is the input feature map and $\mathbf{y}$ the output feature map, T the time index, $\mathbf{W}$ the weights matrix, N the number of input channels, m the output channel, d the dilation parameter, and K the filter size.

2.2 Backblaze Dataset

Our experiments target a real-world hard drive dataset from Backblaze [4], which contains hard drives from many vendors. As [1, 18] pointed out, different disk models require dedicated training since they are characterized by different failure modes. Therefore, we focus on one of the most represented models, the Seagate ST300DM001, comprising data between 2014 and 2017, with a high number of failures, i.e. 1009. The dataset is composed of 90 SMART features collected daily, and the date of failure of the drive, if any. It includes a total of 3828 hard drives of the selected model, resulting in more than one million samples. For each of the 1009 failed hard drives, we assigned the failed label to samples in the last week, as suggested in [18]. With this labeling, the algorithm will learn to predict whether *a given hard drive will fail in the upcoming week*.

Table 1 summarises the main characteristics of the dataset, highlighting the strong class imbalance that is intrinsic in the problem (only 0.60 % of the samples are labeled as failed).

2.3 Related Works

Two main formulations of hard disk predictive maintenance have been proposed in the literature, i.e. Failure Prediction (FP) and Remaining Useful Lifetime (RUL) estimation. In FP, predictive maintenance is addressed as a classification problem. The goal is to predict in advance the occurrence of a failure in the monitored drive. RUL estimation, instead, considers the issue as regression and tries to predict the remaining healthy operating life of the target drive [1].

Table 1. Dataset summary.

Dataset	Backblaze [4]
Model	ST300DM001
# of Hard Disks	3828
# of failed Hard Disks	1009
# of non-failed Hard Disk	2819
Non-failed class samples	1.25M
Failed class samples	7k (0.69%)
# of SMART features (total/after feature sel.)	90/18

Researchers have been attempting to approach both formulations with different machine learning models. In [3], the authors perform FP with Random Forests (RFs), Support Vector Machines, Gradient Boosted Trees, and Logistic Regression, applied to pre-processed SMART features. They also propose to periodically switch algorithms based on their performance in the previous period. RFs are used for fault prediction also in [18], where the authors focus mostly on using online learning to adapt their model to the variance of SMART features over time. Other works have addressed hard disks predictive maintenance using different flavors of deep Recurrent Neural Networks (RNNs), mostly using the Long-Short Term Memory (LSTM) architecture. RNNs have been used for both failure prediction [19] and RUL estimation [14, 2]. The authors of [1] propose to simultaneously perform FP and RUL estimation with a single multi-target LSTM. Interestingly, [2] shows that RFs outperform LSTMs on both continuous and quantized RUL estimation. Moreover, the authors of [14] also experiment with using a standard Convolutional Neural Network (CNN) to estimate the RUL but obtain worse results than those achieved with an LSTM.

3 Methods

In this section, we describe the steps of the pipeline applied to the data to distinguish between failed and healthy hard drives. First, we apply a feature selection stage, followed by the split of train and test data. A re-sampling of the training dataset is then used to manage the strong class imbalance. We conclude with the classification through our proposed Temporal Convolutional Network. Fig. 1 depicts the whole flow of our process, described in the next paragraphs, to assess the status of a hard drive from its SMART values.

3.1 Feature Selection

Before classification, we exclude redundant and irrelevant features from the dataset. This step is crucial to reduce the processing time while also increasing the prediction performance. Therefore, we propose a novel three-step feature selection process to reduce from 90 to 18 the features used during classification:

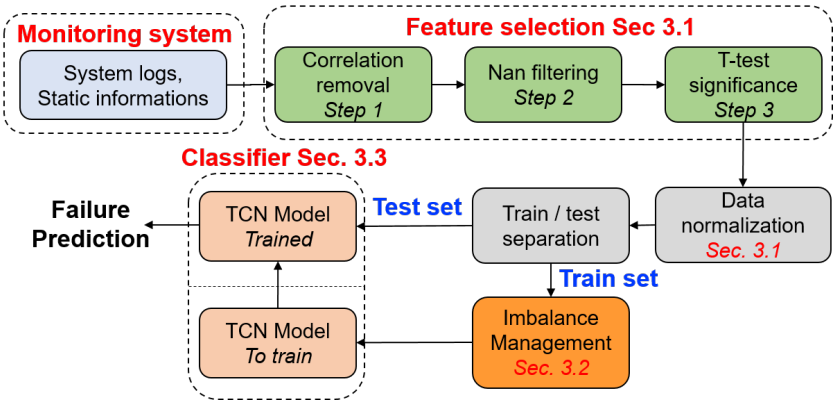

Fig. 1. Pipeline of our method. From raw SMART features to failure prediction.

Step 1. Each SMART attribute is characterized by a *raw* and a *normalized* value. The latter is a vendor-specific normalization of the attribute, computed to obtain a similar distribution of features across different hard drive models. However, for a single model, the *raw* and *normalized* values are strongly correlated (Pearson's coefficient $|r| > 0.99$, p-value < 0.001). Therefore, we removed all *normalized* attributes from our analysis, reducing the features from 90 to 45.

Step 2. We removed all the attributes which have more than 60% of NaN in their values, reducing the features from 45 to 25.

Step 3. To verify the effective predictive power of each attribute, we applied a two-coiled t-test with a significant p-value < 0.001. After the test, we selected only attributes that resulted significant for the distinction between failed and non-failed samples, obtaining the final set of 18 features.

Finally, we normalize the 18 features selected using a Min-Max scaler to avoid bias towards features with larger values:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

3.2 Imbalance Management

As anticipated, the failed class is strongly under-represented in our dataset (0.69%). For this reason, the authors of many prior works [2, 18, 3] propose a random under-sampling of the majority class to reduce the ratio, λ, between the majority and minority class:

$$\lambda = \frac{N_n}{N_f}$$

where N_n is the number of the non-failed samples and N_f of the failed ones. The typical target λ range is $[1, 20]$.

In this paper, we propose to use a more advanced imbalance management technique, i.e. a Synthetic Minority Oversampler (SMOTE). The reader can find details on SMOTE, which are out of the scope of this paper, in [7]. In Sec. 4.2 we show that using SMOTE with a fixed $\lambda = 20$ yields better results compared to those obtained without imbalance management or with random under-sampling.

3.3 TCN Architecture

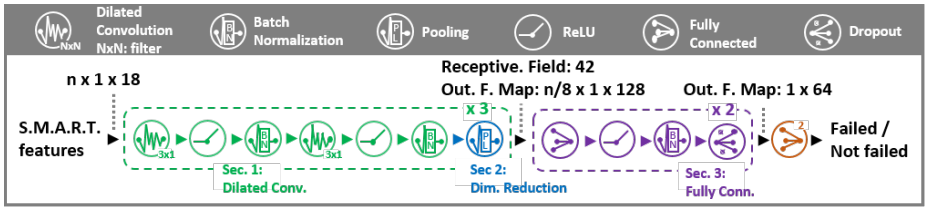

Fig. 2. Topology of the proposed Temporal Convolutional Network.

The TCN architecture used in this work combines dilated 1-D convolutions with max-pooling layers to create a modular structure that progressively reduces the time-length of the input while increasing the number of channels. We designed the network to increase the receptive field through convolutional layers while consuming the time dimension. Note that the architecture is composed of repeated identical *blocks*, as typical in many modern networks, e.g. for computer vision [10, 16]. The whole structure is shown in Figure 2.

The network stacks three convolutional blocks to extract time-relationships between inputs. The three blocks comprise two dilated 1-D convolutions with a 3×1 filter and $d = 2, 2, 4$, respectively, followed by Relu activations and batch normalization. Each block increases the number of channels to 32, 64, and 128, respectively. A final pooling layer halves the time dimension after each block (filter 2×1, stride $= 2$). At the end of the network, three Fully Connected layers with dropout classify the input sequence. A detailed description of the dilated convolutions can be found in [5]. The max-pooling, strided-convolutions, and linear layers are conventional [13].

4 Experimental Results

4.1 Experimental Setup and Metrics

To benchmark our approach, we compared it to the Random Forest of [18] and the LSTM proposed in [1] on the dataset introduced in Sec. 2.2. These two methods are representative of the two most common approaches to hard drives failure prediction in the state-of-the-art. The RF is composed of 30 trees, while

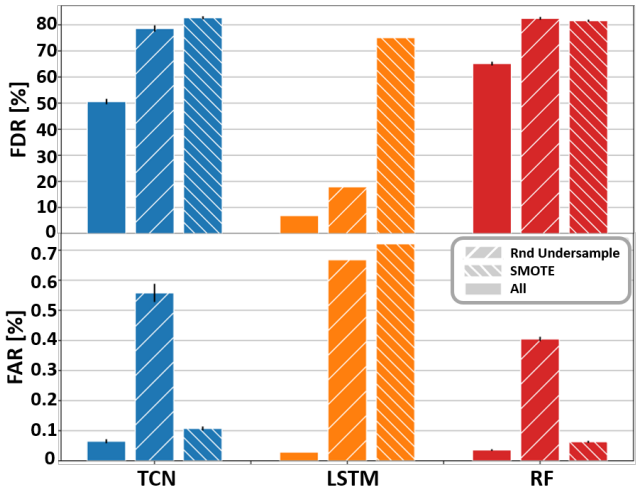

Fig. 3. Effect of different re-sampling methods on the performance of the three classification algorithms.

the LSTM network has a layer of 384 recurrent neurons followed by two fully connected layers. We refer to [18, 1] for further details on the architectures. Note that while the Random Forest receives as an input a single day by construction [18], both the LSTM and the TCN can manage time windows of different lengths. For our experiments, we consider window dimensions between 4 and 90 days.

We use randomly stratified sampling, with 70% of the data used to train all the algorithms and the remaining 30% for testing. On the training dataset, we perform internal validation to search for the best parameters set for the TCN. In detail, we used a batch size of 256 and a learning rate of 0.001, the Adam optimizer, and a decaying factor for the learning rate of 10 every 20 epochs. The maximum number of epochs has been set to 200, using a plateau on the training accuracy of the last 10 epochs as an early stop criterion.

The prediction performance is measured in terms of the following metrics: (1) *Fault Detection Rate* (FDR), i.e. the ratio between the samples predicted as failures and all the real failures, a.k.a. *recall*; (2) *False Alarm Rate* (FAR), i.e. the ratio between the samples wrongly predicted as failures and all the non-failure samples; (3) *Precision*, i.e. the ratio between the correctly predicted failures and all the predicted failures; (4) *F1-score*, i.e. the harmonic mean between precision and recall. All metrics should be maximized except the FAR, which should be minimized. All results are given as mean $\pm$ standard deviation, averaging the last 5 models from the training of the networks, or 5 different Random Forests.

4.2 Comparison of the Sampling Techniques

Our first set of experiments targets the different re-sampling methods used for the management of class imbalance. In Fig. 3, we compare three approaches, namely SMOTE, random under-sampling, and "All" (no re-sampling), using both our TCN and the state-of-the-art methods. For the latter, we apply the same pipeline as for our algorithm, replacing only the final classification step. A window of 30 days has been used in these experiments[5]. As the graph of Fig. 3 demonstrates, not applying any re-sampling techniques results in a very poor FDR from 12% to 28%, lower compared to the application of the random under-sampling/SMOTE for all the three classification algorithms. Moreover, although the random under-sampling guarantees a good FDR, it causes the FAR to increase by a factor of 9×/20× compared to the previous case. On the other hand, SMOTE maximizes the FDR for the TCN and the LSTM (for the RF, the FDR is less than 1% lower than with random-undersampling), while achieving a FAR on the TCN and the RF only 1.65×/1.75× higher compared to not applying re-sampling. Overall, the F1-score of the TCN, LSTM, and RF using SMOTE increases of 20.48%, 38.5%, and 9.08% compared to not re-sampling and of 21.38%, 43.5%, and 17.16% compared to the random under-sampling.

Therefore, we use the SMOTE to re-sample our training dataset in all the following experiments.

4.3 Algorithms Comparison

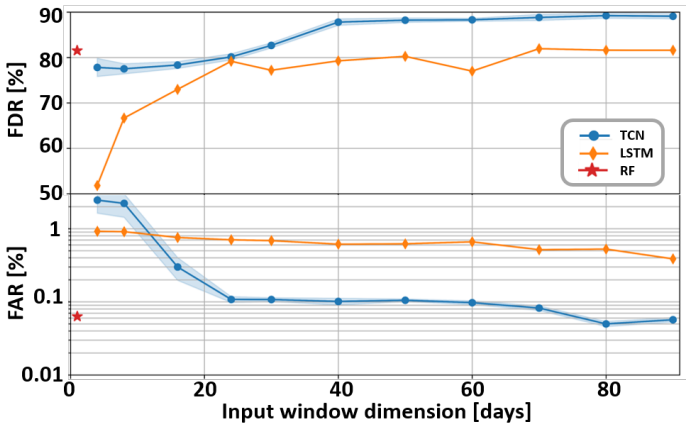

Fig. 4. LSTM and TCN performance for different input window sizes.

[5] For this input window size, the TCN achieves a FDR similar to the RF, but the following experiments show that the TCN FDR improves for longer inputs.

Fig. 4 shows the performances of LSTM and TCN with respect to the length of the input data used as input. Note that the RF has been executed on single day input samples as in its original work [18] since it intrinsically doesn't support the analysis of time series. Both the LSTM and our TCN demonstrate better performance by using more days in the input signal. However, for both FDR and FAR, the TCN is capable of taking more advantage from a longer input window. In particular, compared to an input window of size 24, on which the LSTM and the TCN achieve a similar FDR (79.21% vs. 80.23%), the LSTM increases its FDR by less than 2.5% when using a window of 90 days, while the TCN reaches FDR = 89.1%, with an improvement of 8.87% compared to the 24 days case.

Table 2 summarises the comparison of the three methods: the TCN proposed, the LSTM of [1], and the RF presented in [18]. Note that all three algorithms exploit the same steps of pre-processing, differing only in the classification step. When using 90 days of history, the TCN surpasses the competitors by 7.49% (RF) and 7.55% (LSTM) in FDR, achieving slightly lower FAR (0.052 vs. 0.063). We use the F1-score to give an idea of the overall comparison between the three methods. Our TCN achieves an average F1-score of 90.05%, compared to the 85.52% obtained by the RF and the 65.5% of the LSTM (the reached performance in our work are almost equal to the ones presented by the authors of the referenced papers). Note that both [1, 2] already showed that RF often outperforms LSTM for this kind of task.

These better metrics would directly translate into monetary savings, e.g. for cloud storage providers. Focusing on the FDR, i.e. the number of correctly predicted hard drives failures, we can see that our algorithm mispredicts only 1 failure over 10, compared to the 2 errors over 10 made by the other methods. Considering a realistic cloud storage using a Redundant Array of Independent Disk (RAID) [8] with hot-spare disks, we can link the FDR to a reduction in the number of spare drives with a 1:1 relationship, e.g. 80% FDR allows to reduce the number of the spare disks by 80%. Hence, halving the failure mispredictions implies decreasing by a factor of 2× the extra cost for the redundancy in the data center.

Table 2. Performance comparison of the three algorithms.

	RF [18]	TCN – 90 d	LSTM [1] – 90 d
FDR [%]	81,55 ± 0,40	89,10 ± 0,57	81,61 ± 0
FAR [%]	0,063 ± 0,003	0,052 ± 0,004	0,387 ± 0
Precision [%]	89,92 ± 0,47	91,00 ± 0,69	54,8 ± 0
F1-Score [%]	85,52 ± 0,22	90,05 ± 0,13	65,5 ± 0

4.4 Model Size Exploration

Finally, inspired by the results in [10], we analyze the impact of the size of the proposed TCN by applying a *width multiplier* and a *depth multiplier* to

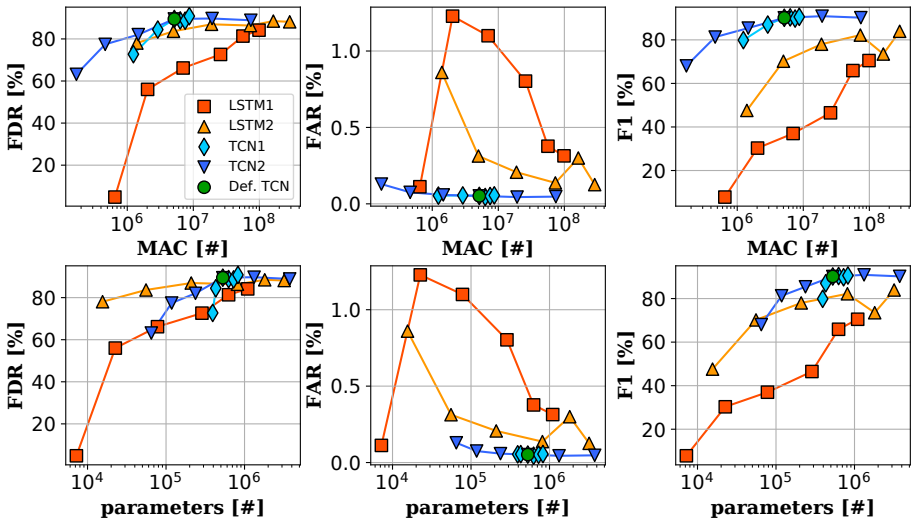

Fig. 5. FDR, FAR and F1 for varying TCN and LSTM model sizes.

increase and decrease the number of channels per layer and the number of layers. Consequently, we increase or reduce both the network size and the number of operations performed for each classification.

Specifically, we regulate the depth of the network considering six models with {1, 2, 3, 4, 5, 6} convolutional blocks respectively (labeled TCN1 in Fig 5). Convolutional blocks after the third are identical to the first three described in Section 3.3 and use 128 channels, $d = 4$, and full padding, but do not include any time dimension reduction. Then, we also multiply the number of channels of each layer by {0.125, 0.25, 0.5, 1, 2, 4}, keeping the default depth of 3, creating six additional models (TCN2 in Fig. 5).

For comparison, we also explore variations of the state-of-the-art RNN architecture using either one or two stacked LSTM layers and {32, 64, 128, 258, 384, 512} hidden layer dimensions. In Fig. 5, we label single-layer models LSTM1 and two-layer models LSTM2. We train all TCNs and LSTMs with identical hyperparameter settings. We do not report any exploration for the Random Forest, since increasing the number of trees does not improve the performance.

Fig. 5 shows the FAR, FDR, and F1 score of the twenty-four models considered in this exploration. On the x-axis, the plots report either the number of Multiply-and-Accumulate (MAC) operations or the number of parameters of the corresponding models. The first conclusion that can be drawn from this exploration is that the TCN with channel multiplying factor (CMF) = 1 and three layers (the default structure presented in the paper, shown as a green dot in the figure) is the smallest model that saturates performance, and therefore represents a good trade-off point. Increasing the model size further does not cause a dramatic increase in performance. We obtain the best performance using CMF = 2, reaching an F1 score of 90.9 %, only 0.8 % better compared to the base

architecture. We also note that with progressively bigger architectures, the FAR is almost constant (min 0.44, max 0.55), while we can observe an increase of the FDR of nearly 30%. Hence, we conclude that small TCNs are sufficient to extract features that distinguish well the *not-failing* category. On the other hand, larger and deeper networks are needed to identify *failing* hard drives correctly, due to the wide range of possible failure modes.

Importantly, for a similar number of MAC operations or parameters, the proposed TCN versions almost always outperform the LSTMs. In particular, the top-right plot of Fig. 5 shows that LSTM architectures are never Pareto optimal in terms of F1-score versus the number of MAC operations. When considering the number of parameters (bottom-right plot), the trend is similar; the only exception is represented by architectures with less than 55k parameters, which, however, reach a F1-score lower than 70%, insufficient for an accurate monitoring system.

5 Conclusions

We have shown that TCNs can outperform state-of-the-art methods for hard disk failure prediction. This improvement is mostly thanks to their excellent long-term memory, which allows them to take advantage of long input time-windows. We have also shown that, for TCNs, applying SMOTE before training helps to improve the FDR while keeping FAR contained. In our future work, we will use TCNs also for RUL estimation, and experiment with unsupervised training to tackle the class imbalance problem. Moreover, this work has shown that small and low-complexity TCNs can still achieve good performance (e.g. the model with CMF=0.25 in Fig. 5 has $32\times/160\times$ fewer parameters/MACs than the biggest TCN, while achieving a decent F1 score of 80.3%, and a FAR $<0.1\%$). This result could allow the execution of TCN-based inference in emerging ultra-low-power MCU architectures, which despite being performance- and memory-limited, offer specific HW and SW features for the execution of tiny ML models [9, 6, 11]. Our result, combined with these new architectures, could allow the embedding of the TCN in the HDD controller, enabling a scenario in which each hard drive autonomously provides fault prediction alarms to the system. Investigating this scenario will also be part of our future work.

Acknowledgments This work has been partially supported by the H2020 project IoTwins (g.a. 857191).

References

1. Aggarwal, K. et al: Two Birds with One Network: Unifying Failure Event Prediction and Time-to-failure Modeling. CoRR **abs/1812.0** (2018), http://arxiv.org/abs/1812.07142
2. Anantharaman, P. et al: Large Scale Predictive Analytics for Hard Disk Remaining Useful Life Estimation. In: 2018 IEEE BigData Congress. pp. 251–254

3. Apiletti, D. et al: iSTEP, an Integrated Self-Tuning Engine for Predictive Maintenance in Industry 4.0. In: 2018 IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom. pp. 924–931
4. Backblaze: Backblaze dataset (2019), https://www.backblaze.com/b2/hard-drive-test-data.html
5. Bai, S. et al: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
6. Burrello, A. et al: Work-in-Progress: DORY: Lightweight Memory Hierarchy Management for Deep NN Inference on IoT Endnodes. In: 2019 CODES+ISSS, pp. 1–2.
7. Chawla, N.V. et al: Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research **16**, 321–357 (2002)
8. Chen, P.M. et al: Raid: High-performance, reliable secondary storage. ACM Computing Surveys (CSUR) **26**(2), 145–185 (1994)
9. Garofalo, A. et al: PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **378**(2164), 20190155 (feb 2020).
10. Howard, A.G. et al: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017), http://arxiv.org/abs/1704.04861
11. Jahier Pagliari D. et al: Energy-Efficient Digital Processing via Approximate Computing. In: Bombieri N., Poncino M., Pravadelli G. (eds) Smart Systems Integration and Simulation. Springer, Cham (2016).
12. Lea, C. et al: Temporal convolutional networks for action segmentation and detection. 2017 IEEE CVPR pp. 1003–1012
13. LeCun, Y. et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**11, 2278–2324 (1998).
14. Lima, F.D.S. et al: Remaining Useful Life Estimation of Hard Disk Drives based on Deep Neural Networks. In: 2018 IJCNN. pp. 1–7
15. Manousakis, I. et al: Environmental Conditions and Disk Reliability in Free-cooled Datacenters. In: 14th {USENIX} {FAST} 2016. pp. 53–65.
16. Sandler, M. et al: Mobilenetv2: Inverted residuals and linear bottlenecks. In: 2018 IEEE CVPR, pp. 1–14.
17. Schroeder, B., Gibson, G.A.: Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In: 5th USENIX FAST, pp. 1–16.
18. Xiao, J. et al: Disk Failure Prediction in Data Centers via Online Learning. In: 47th ACM ICPP, 2018, pp. 1–10.
19. Xu, C. et al: Health Status Assessment and Failure Prediction for Hard Drives with Recurrent Neural Networks. IEEE Transactions on Computers **65**(11), 3502–3508 (2016).