What Does Pruning Actually Buy? Memory and Inference Time in Predictive Maintenance Models

Daniel Grillmeyer Sükrü Turhal Samuel Kounev
University of Würzburg
{daniel.grillmeyer, samuel.kounev}@uni-wuerzburg.de

Abstract

We evaluate unstructured and structured pruning for resource-efficient inference of two Predictive Maintenance (PdM) Deep Learning (DL) models. Prior work investigates only a single pruning technique, few pruning ratios, and no standardized metrics. We re-implement a Deep Convolutional Neural Network (DCNN) achieving State-of-the-Art results, apply both methods at ratios of 10% to 90% on the NASA Turbofan dataset, and evaluate the Root Mean Squared Error (RMSE) and S-Score alongside computational load. Unstructured pruning is less harmful to the predictive performance but does not improve resource efficiency at all. Structured pruning reduces memory consumption by up to 87.5% and Multiply-Accumulate Operations (MACs) and Floating-Point Operations (FLOPs) to around 4%, while 20 epochs of fine-tuning restore almost all lost predictive performance. At a batch size of 512, the pruned models nonetheless remain above the RAM of typical industrial microcontrollers.

1 Introduction

Efficiency is the common denominator for a lot of modern industrial problems. If a machine suddenly breaks down, production halts, and a loss of efficiency always results in a loss of money, so running a machine until its failure is not a feasible option. Scheduled, regular check-ups with preventive maintenance waste resources instead: an unnecessary check-up wastes money, while a schedule that misses the point where the machine starts to break down still leads to a sudden outage and even greater costs.

The third and best option is PdM, which uses sensor readings and data-driven models to predict the Remaining Useful Life (RUL) of machinery, that is, the estimated time or operational cycles left before a machine fails and requires maintenance. We choose a DL approach for its potential generalization capabilities across multiple different machine types: given a machine's previous sensor readings, a DL model learns a mapping from those to the correct RUL value, so that an optimal time for maintenance can be determined. DCNNs are particularly well-suited for this task, as they learn spatial and temporal patterns directly from raw input signals instead of manually engineered features. However, their high parameter count brings high computational demands, which makes it difficult to deploy them for real-time inference on resource-constrained edge devices close to the machine, the setting that motivates this study. Pruning is designed to counteract this drawback, ideally leading to less computational load and in some cases even to better predictive performance.

2 Background & Related Work

Two main categories of pruning techniques exist. Unstructured pruning identifies the least important weights and zeroes them out, leading to sparse weight matrices. Structured pruning removes entire structures such as neurons or convolutional filters, which can disrupt the learned feature representations [7, 8].

Li et al. [3] presented a DCNN that extracts features from time-windowed sensor data, outperforming traditional machine learning methods [6] but at a high computational cost that challenges edge deployment; we adopt it as our baseline. Athanasakis et al. [4] apply unstructured pruning at a fixed 25% rate on the same dataset and deploy the models on a microcontroller, but find that only quantization effectively reduces model size and inference time. Pandey et al. [9] use L2 norm-based structured pruning at a fixed 15% ratio without fine-tuning, achieving significant size reduction with a 2% average accuracy drop. Liu et al. [5] prune the same DCNN with L1 norm at 10%, 20% and 30% with fine-tuning, but do not specify which Turbofan subset is used and omit the S-Score metric.

Prior works typically apply low pruning ratios, which leaves the trade-offs between model compression, resource efficiency and predictive performance unexplored. Additionally, some related work lacks standardized evaluation metrics, so their results are not directly comparable. In contrast, we apply two pruning methods, explore higher pruning ratios to better characterize performance degradation, and employ both RMSE and S-Score metrics for comprehensive evaluation.

3 Experimental Setup

3.1 Model & Dataset

We re-implement the DCNN proposed by Li et al. [3]: five convolutional layers, a dropout layer with a rate

of 0.5, and two fully connected layers, the first with 100 neurons. The first four convolutional layers use filters of size $(10, 1)$ with ten output channels each, the fifth a $(3, 1)$ kernel reducing to one channel. The input is a time window of size N_{tw} over 14 sensor readings. We use the C-MAPSS Turbofan dataset [1], dropping the seven of its 21 sensors that do not vary over time, and scaling the rest with min-max normalization to a range of [-1, 1]. Two sub-models are evaluated: the big model on FD001 with $N_{tw} = 30$ and the small model on FD004 with $N_{tw} = 15$.

3.2 Pruning Configuration

Both methods use the L1-norm to evaluate model structures. Unstructured pruning targets the weights of the convolutional layers, while the structured approach targets the channels. The fully connected layers are not pruned, as their operations are computationally insignificant compared to the convolutional operations. We apply ratios of 10% to 90% in steps of 10%, and fine-tune afterwards for 20 epochs with the Adam optimizer [2] at a learning rate of 0.001. Each configuration is run 10 times; inference time is the median of 100 measurements taken after 1,000 warm-up measurements.

3.3 Metrics

Predictive performance is measured with the RMSE and the S-Score, the Turbofan dataset-specific measure which sums an exponential penalty over all test engines that grows faster for late than for early RUL predictions [3]; for both metrics, lower is better. Deployability is measured as the storage size and the memory consumption in MB plus the inference time in ms, and complexity as the parameter count, the MACs and the FLOPs. Memory is measured with the built-in PyTorch Profiler over one forward pass of a batch of 512 samples, so it includes all intermediate activations; the weights alone take up far less. Inference time is measured as wall-clock latency on the same input.

4 Results

All results are averaged over ten runs and reported as mean ± standard deviation. Because pruning is applied deterministically to a single fixed baseline checkpoint, the metrics measured without fine-tuning show no run-to-run variance; for those we report the mean only. All measurements were taken on a 10 GB MIG slice of a server with two AMD EPYC 7763 64-core CPUs at 2.45 GHz, 256 GB of RAM and two NVIDIA A100 80 GB GPUs, under Ubuntu 22.04.5 LTS.[1]

4.1 Predictive Performance

Our re-implementation reaches an RMSE of 12.59 and 24.55 and an S-Score of 254.6 and 5,043.1 for the big and the small model, compared to 12.61 and 23.31 and 273.7 and 12,466 reported by Li et al. [3].

Without fine-tuning, the RMSE degrades with increasing pruning ratio for both methods and models, but unstructured pruning is clearly less harmful than structured pruning. After unstructured pruning at a ratio of 20% both models are almost unaffected at 13.13 and 24.72, whereas structured pruning at the same ratio yields 16.95 and 57.48. The maximum RMSEs are 44.51/54.87 for unstructured pruning, both at 90%, and 45.70/58.58 for structured pruning at 60%/80% for the big/small model. Structured pruning degrades far earlier: already at 10% it reaches 16.79 and 46.23, while unstructured pruning still stays at 12.73 and 24.45. At 10% unstructured pruning the small model even outperforms the original by 0.10 RMSE points, showing that pruning can sometimes improve model performance.

Fine-tuning recovers most of this degradation. With 20 epochs the RMSE stays within 1.71 points of the unpruned baseline for both models, for structured and unstructured pruning alike, even at a ratio of 90%, where the big model reaches 13.80 ± 0.22 for unstructured and 14.27 ± 0.16 for structured pruning, and the small model 25.80 ± 0.24 and 26.03 ± 0.40. The S-Score follows the same trends with stronger degradation and more variance: at 90% it is 378.5 ± 36.8 and 407.5 ± 34.1 for the big model and 5,610.6 ± 785.0 and 5,982.8 ± 805.2 for the small model.

4.2 Resource Efficiency

Fine-tuning has no effect on the deployability and complexity metrics, so we evaluate the pruned models without it. Unstructured pruning improves none of them: it removes no parameters or structures, and the big model's inference time stays flat at (13.18±0.02) ms across every ratio. PyTorch even adds a "mask" buffer recording which weights are zeroed, increasing the storage size of both models by 11.9 KB. Structured pruning *saves* the same negligible 11.9 KB at 90%, since it can only remove 3,123 of the 3,171 convolutional parameters, while the untouched first fully connected layer dominates the storage size.

As Figure 1 shows, structured pruning reduces the memory consumption of both models by 87.5%, following an approximately linear degradation of −1.61 and −0.92 MB per 1% pruning for the big and the small model. At 90%, the big model consumes 20.87 MB and the small model 11.44 MB, so the absolute reductions are 146.18 MB and 79.73 MB, respectively. Inference time decreases as well, from (13.17 ± 0.01) ms to (5.07 ± 0.01) ms for the big model and from (12.19 ± 0.01) ms to (4.94 ± <0.01) ms for the small model, a speed-up of 2.60× and 2.47×. The decrease is not smooth, however: the big model drops sharply between 40% and 50%, from (11.93 ± 0.01) ms to (6.77 ± 0.01) ms, and declines only slowly afterwards, and the small model's measurements are non-monotonic

[1] Our code and data are persistently available at `https://doi.org/10.5281/zenodo.23469324`.

at low ratios. This suggests that the measured latency is dominated by kernel launch and scheduling overhead rather than by the remaining arithmetic, so the absolute values are device-specific.

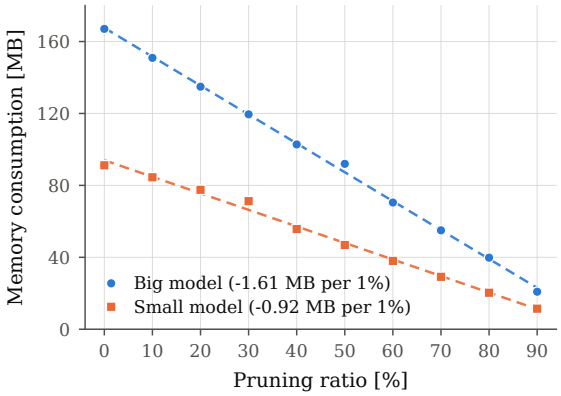

Figure 1: Memory consumption across structured pruning ratios for both models.

4.3 Complexity

Unstructured pruning has no effect on the parameter count, the MACs or the FLOPs, because it only zeroes out the weights and does not physically remove any parameters. Structured pruning has a clear effect on all three, but the parameter count in itself is not a good indicator of complexity and thus deployability: it was only reduced by 3,123 for both models, starting at 42,572 and 21,572 respectively, because the 39,300 and 18,300 weights of the first fully connected layers are never pruned. The MACs fall from 692.77/345.48 MMACs to 29.33/13.95 MMACs and the FLOPs from 1.3943/0.6953 GFLOPs to 59.71/28.42 MFLOPs for the big/small model, leaving around 4% of both operations. Most of the complex operations are indeed in the convolutional layers; pruning the first fully connected layer is left for future work.

5 Discussion

Structured pruning brought the memory consumption down to around 21 and 11 MB for the big and the small model with reasonable predictive performance after fine-tuning. The most widespread MCUs for industrial PdM are of the STM32 series, but even their most performant one, the STM32N6, built for edge AI, has only 4.2 MB of RAM. At a batch size of 512, our pruned models therefore exceed the RAM of lower-end devices; single-sample inference may fit, but we did not measure it in this study.

Unstructured pruning creates a multitude of sparse matrices dominated by zeroes. In theory this should improve calculation times, but in practice the hardware accelerators do not automatically optimize operations on these matrices, so the computational costs are equal to normal non-sparse matrices. Athanasakis et al. [4] observed the same at a ratio of 25%, and we assumed that this low ratio was not enough; even at 90%, however, our results did not show improvements.

Four threats to validity remain. Memory and inference time were measured with a batch of 512 samples on an NVIDIA A100 GPU that parallelizes batch calculations. No physical deployment was conducted, so our results are theoretical projections, and the unpruned fully connected layers bottleneck the storage size. Finally, we evaluate a single DCNN architecture on two subsets of the C-MAPSS dataset, so our results may not transfer to other PdM datasets or architectures.

6 Conclusion

We evaluate two pruning techniques in their ability to make two differently sized PdM models more resource-efficient. Only structured pruning yields results: it reduces the computational load of both models immensely, whereas unstructured pruning yields no reduction at all. We tested ratios of up to 90%, whereas most pruning research in the field of PdM sets their limits at around 30% to 40%, and while the predictive performance did degrade strongly—for structured pruning already below those thresholds—fine-tuning for only 20 epochs brought it back to within 1.71 RMSE points of the unpruned baseline. Future work might implement unstructured pruning in a way that yields computational efficiency, prune the fully connected layers, and physically deploy the pruned models to edge devices.

References

[1] NASA. *CMAPSS Jet Engine Simulated Data*. 2008.

[2] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *Proc. ICLR*. 2015.

[3] X. Li et al. “Remaining useful life estimation in prognostics using deep convolution neural networks”. In: *Reliab. Eng. Syst. Saf.* 172 (2018).

[4] G. Athanasakis et al. “TinyML-based approach for remaining useful life Prediction of Turbofan Engines”. In: *Proc. IEEE ETFA*. 2022.

[5] H. Liu, P. Song, Y. Qie, et al. “Real-time Prediction Method of Remaining Useful Life Based on TinyML”. In: *Proc. IEEE Int. Conf. RCAR*. 2022.

[6] O. Serradilla, E. Zugasti, J. Rodriguez, et al. “Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects”. In: *Applied Intelligence* 52.10 (2022).

[7] S. Vadera and S. Ameen. “Methods for pruning deep neural networks”. In: *IEEE Access* 10 (2022).

[8] G. Fang et al. “Depgraph: Towards any structural pruning”. In: *Proc. IEEE/CVF CVPR*. 2023.

[9] R. Pandey et al. “Towards deploying dnn models on edge for predictive maintenance applications”. In: *Electronics* 12.3 (2023).