SpaceOps-2023, ID # 274

Spacecraft On-board Anomaly Detection: computational constrained Machine Learning approaches

Salvatore Cognetta[a]*, Carlo Ciancarelli[a], Francesco Corallo[a], Eleonora Mariotti[a], Antonio Leboffe[a]

[a] *Thales Alenia Space Italia S.p.A.*
* Corresponding Author

Abstract

Modern spacecraft are complex systems composed of several advanced equipment. Each of the on-board units produces dozens of telemetry series in order to make the system observable by the ground operators and engineers, who periodically downlink and store the telemetry, allowing the monitoring of the on-board events, the equipment health status, and the on-board anomalies investigation. Moreover the monitoring activity is becoming increasingly challenging as large constellations are becoming more common, and the need for automated monitoring solutions is growing.

In this work we exploit the use of an AutoEncoder Neural Network for the fault/anomaly detection in a satellite subsystem telemetry data, taking care of all the steps of the optimization process required for a direct implementation on a low performance hardware device. We propose the implementation of a quantization scheme that allows inference to be carried out using integer-only arithmetic (int8), which can be implemented more efficiently than common floating point inference on integer-only hardware. As we will explain, this process reduce the model complexity and memory footprint. Moreover, to reduce model complexity, weight pruning of the AutoEncoder is applied, obtaining a reduction in the number of needed Multiply and Accumulation operations point of view (MADD).

It will be shown how the reduction of the ML-based anomaly detection system is capable of efficiently detect an anomalous pattern both in its original architecture and in its quantized one.

Keywords: Spacecraft, Anomaly Detection, Machine Learning, Quantization, Pruning

1. Introduction

Spacecraft Prognostic and Health Management (PHM) systems, also referred to as Fault Detection, Isolation and Recovery (FDIR), are a key component of any space mission design. This system greatly affects the spacecraft's availability, reliability, and safety goals. They use a variety of sensors and algorithms to monitor the health of the spacecraft and its subsystems. Fault and anomaly detection systems are needed also to alert space operations engineers of anomalous behaviour and prevent significant failures.

In this work we focus on the Fault Identification (FI) part, while the isolation and recovery actions is left to the traditional FDIR system, generating therefore an enhanced-FDIR on-board system that consider the Machine Learning (ML) model in the loop.

Current health monitoring systems require an expert knowledge to be developed and maintained, in order to comply the increasing scale and complexity of spacecraft coupled with the difficulty of providing cost-effective ground operations. The state of the art anomaly detection methods for spacecraft telemetries consist of a deterministic monitoring process, called "Out-Of-Limits" (OOL), verifying whether the monitored sensor telemetry values are within pre-defined limits or stray outside them. These approaches have different limitations, alongside the costly expert knowledge to define and update the nominal ranges/threshold values and tables for each telemetry sensor, there is the need of ongoing manual analysis of telemetry data which can be translated in workload for space operations engineers.

The development of such systems is a challenging task, in fact beyond the abovementioned problems, the complexity of the system and the need for real-time performance shall be considered. In this context, ML techniques have been proposed as a viable solution to detect anomalies in space systems. ML algorithms can be used to detect anomalies in telemetry data by learning the normal (or nominal) behaviour of the system and then flagging any deviation from it. The most common ML techniques used for anomaly detection are unsupervised or semi-supervised learning algorithms, such as Support Vector Machines (SVMs) [1] and Artificial Neural Networks (ANNs).
These algorithms are trained on a dataset of normal behaviour and can then be used to detect anomalies in new data. Unsupervised learning algorithms, such as clustering algorithms, can also be used to detect anomalies. These algorithms group similar data points together and then flag any data points that do not fit into the clusters as anomalous.
In addition to ML techniques, other methods have been proposed for anomaly detection in space systems. These include rule-based systems, which use predefined rules to detect anomalies; statistical methods, which use statistical tests to identify outliers; and hybrid systems, which combine different methods for improved accuracy. No matter what technique is used for anomaly detection in space systems, it is important that the system is able to accurately identify both false positives (normal behaviour flagged as anomalous) and false negatives (anomalous behaviour not flagged).

Another main step for the exploitation of ML in spacecraft enhanced-FDIR is, obviously, the direct implementation on space processors. This kind of processors has different HW limitations, that shall be considered in the classical ML Pipeline. In fact, to deploy the models we will take care of the resource-constrained memory, the integer-only arithmetic (int8 precision) opposed to the common floating point arithmetic and the other limitations (better described in Section 2.2) that these systems place on.
To face these problems we have to consider the Quantization: it is a process used to reduce the size and computational complexity of ML models, without sacrificing too much accuracy. It involves replacing the high precision weights and activations of the model with low-precision representations, such as 8-bit integers instead of 32-bit floating-point numbers. The quantized model is then re-trained, fine-tuned, or calibrated to recover accuracy lost during the quantization process. This allows the deployment of the model on resource-constrained devices, such as mobile phones and embedded systems (as in our case), with reduced memory and computational requirements.
Quantization also reduces the number of parameters in a model, which can lead to a smaller model with lower memory footprint and power consumption [2], crucial for spacecraft edge computing deployment.

Another method for ML models complexity reduction is pruning: it is a technique used in the field of deep learning to reduce the number of parameters in a neural network, thereby reducing its computational complexity and memory requirements. It is a well-established method for improving the efficiency of neural networks and has been widely used in various applications, including computer vision and natural language processing.
In the pruning method of neural networks, a pre-trained network is analyzed to identify the neurons that have the least impact on the overall performance of the network. These neurons are then removed from the network, resulting in a smaller network that still preserves the accuracy of the original network.

Thales Alenia Space Italia (TAS-I) Space Segment Operations unit decades experience in supporting the operations and health monitoring of spacecraft constellations confirms the need to evolve traditional approach with innovative monitoring systems, capable of automating the analysis of large amounts of telemetry data, and alert the ground operators and engineers of unusual or novel patterns worthy of a deeper human analysis, reducing the workload and spacecraft operational downtime, and improving the quality of the service. Moreover the telemetry data downlinked from the spacecraft contains valuable information which can be exploited to model the on-board units behaviour, allowing to predict the future behaviour evolution.

In this paper, we aim to provide an implementation of state-of-the-art ML model, namely AutoEncoder (AE) [3], for anomaly detection, its architecture and the required steps for low performance hardware implementation (e.g. quantization and weight pruning).

1.1 Related Works

AutoEncoder models have recently emerged as a powerful tool for anomaly detection. These models are deep learning algorithms that have the ability to identify patterns in large datasets and can be trained to reconstruct data in an efficient manner.
It is a type of neural network that is trained to reconstruct its input. The idea behind it is to learn a compressed representation of the input data, called the encoding, and then to use that encoding to reconstruct the original input data. This is done by training the network to minimize the difference between the input and the reconstructed output. The encoded data is a compact "summary" or "compression" of the input, also called the latent-space representation.
By reconstructing data, the model can learn to identify not-nominal patterns and deviations from those patterns that may indicate anomalies.

The framework described in [4] proposed an approach for fault detection and prediction based on AutoEncoder model, using Long Short Term Memory (LSTM) Neural Networks for the Encoder and Decoder modules, in a steel production case study. The AE model was trained using a dataset consisting only of nominal time-series sequences corresponding to a given state (semi-supervised learning). The scope of this training it to compute the reconstruction error, derived from the AE itself, meaning the error between the original input data and the reconstructed one after the encoding and decoding phases of the AE. Hence, anomalies are identified if this reconstruction error (or anomaly score) increases significantly from its nominal behaviour value. Performance results obtained with the prototype demonstrated that unnecessary preventive maintenance actions could be reduced, therefore decreasing the cost of maintenance operations.

Another example of the applicability of AE model in anomaly detection is depicted in [5], where an AE has been trained on spacecraft telemetry data to identify synthetic and injected anomalies in the dataset. Also in this case, the AE is trained using only nominal samples, in order to identify a nominal reconstruction error, exploited in inference time: in short words when the telemetry deviates from it, the telemetry is identified as anomalous. The results obtained shows that the ML model is capable to identify anomalies whit increasing order of complexity.

On the other hand, Jacob et al. [6] proposed a Quantization and Training procedure that allows inference using integer-only arithmetic on integer-only HW. The proposed quantization scheme improves the trade-off between accuracy and on-device latency. The improvements are significant, using Large Networks like ResNets [7] and InceptionV3 [8].

Moreover, as described in [9], moving from floating-point representations to low-precision fixed integer values can reduce the memory footprint and latency by a factor up to 16x. The work by Nagel et al. [10] points out this performance improvement describing the state-of-the-art quantization algorithms, taking into consideration two main classes of algorithms: Post-Training Quantization (PTQ) and Quantization Aware Training (QAT). The first apply quantization after the training, where quantization parameters are calculated based on sample calibration data, while the second simulate quantization during the training so that the quantization parameters (weights and activations) can be learned together with the models.
In this work we focus on QAT, which yields at higher accuracies than PTQ method [11] for small models.

1.1.1 Paper structure

The paper is organized as follows. In Section 2.1 a background on the satellite FDIR mechanisms is provided, as an introduction to the proposed operational use case. Section 2.2 gives a description of hardware devices for spacecraft On Board Computer, giving a view on current state-of-the-art processors and the next gen one that will increase power and on-board capabilities. Section 3 is devoted to the description of the dataset taken into consideration. Section 4 is dedicated to the description of the ML model implemented and the optimization processes applied to the original model. Finally, in Section 5 the obtained results are described where the memory footprint and the MADDs of each method are estimated.
Finally, some conclusions are drawn in Section 6.

2. Spacecraft monitoring and system requirement

2.1 Fault Detection Isolation and Recover - FDIR

The satellite FDIR logic has the aim of detecting, isolating and recovering faults at unit, subsystem or equipment level. It is actually based on a hierarchical architecture trying to confine failures at the lower FDIR levels to minimize outages and provide system availability. This strategy is implemented within the Avionic SW (ASW) in the form of several predefined tables containing selected monitoring items and relative recoveries. These tables are designed according to experience, then one of the most important disadvantage is that they are not able to detect failures are not provided by the designer. Its functionality foresees that a proper set of selected parameters computed by the ASW does not exceed the predefined operational thresholds. After a confirmation time ("filter" configuration data), the detection of the violation of a monitoring criterion triggers a recovery action. Confirmation time is generally introduced to eliminate spurious or transient events that don't affect the system; however, the deterministic nature of this strategy based on fixed thresholds and time limits constitutes a limit for on-board failure detection.
The FDIR performances could be potentially affected by this limited nature of a table-driven approach. In fact, the software logic is not flexible and it is not able to recognize any type of failure, but only those ones that are expected by design and for which the parameters have been selected to be monitored. Furthermore, it does not guarantee any preventive maintenance. In general, the highest level of the FDIR hierarchy is in charge to the Ground Control Centre (GCC). The GCC is able to send telecommands in order to enable/disable the on-board autonomous FDIR operations, or for setting the FDIR configuration parameters and logics. When the satellite is affected by unexpected recoveries due to lack of design or prediction capabilities, GCC shall investigate about the anomaly causes and it could take long times to restore the satellite functionalities. For this reason, the introduction of on-board ML approaches for FDIR could overcome these problems, especially in identifying and isolating failures at the lowest level possible (equipment level) thus fostering equipment/software reuse, mission availability and autonomy. In fact, the ML algorithms could be able to analyze a big amount of on-board available data and recognize failures not foreseen by design, or could react faster than the fixed confirmation time.
In the fields of FDIR design space hardware limitations shall be taken into account. Space computers are really much less powerful than the terrestrial ones. On-Board Computers are characterized by space qualified components, since hardware shall be reliable to space environment (for example, ionizing radiation), and at the same time they are very limited from computational capabilities point of view. They are characterized by maximum volatile and non-volatile memories constraints, maximum stack of any algorithm function etc., i.e. all characteristics relevant for ML algorithms usage. In order to use the ML algorithms on-board, it should be also necessary to investigate about the possibility to optimize software coding and reduce the computational needs, paying attention to the use of external AI and ML libraries that should be no hardware dependent or adaptive to space software compilers. In addition, the flight software is generally coded according to quality rules and always paying attention to safety. This means also the ML algorithm implemented on-board shall satisfy this coding standard.

2.2 Spacecraft OBC limitations

The hardware used for spacecraft OBC is subjected to a number of constraints due to space-specific factors. Radiation tolerance is the most important factor since the measures adopted to mitigate its effects (e.g. lowering clock frequency, introduction of redundancies) directly affects the final performances. Another important factor is the memory footprint of the application. The latter can change according to the hardware platform used and it has to be carefully considered since mass memory can be scarce and, even more important, the bandwidth for any updates can be limited. Finally the status of the development tools available should be carefully evaluated: the presence of license-free and long-term supported libraries, frameworks, and Software Development Kit can considerably speed up and simplify the development of applications. For further details about this topic see [12], [13] and [14].

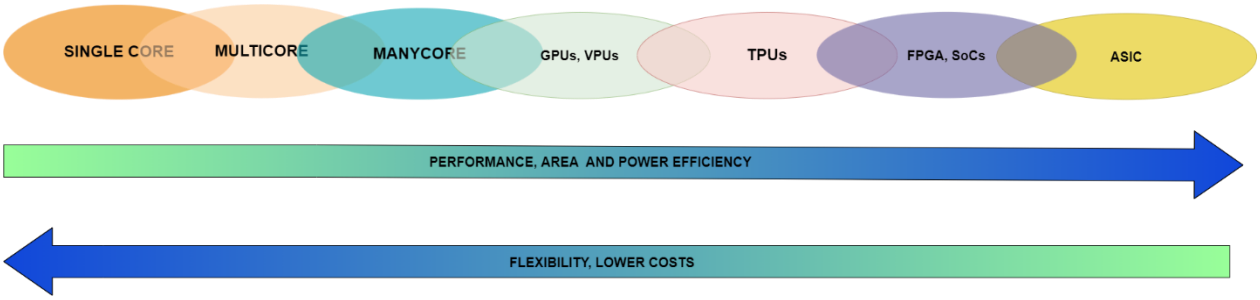
In the following the hardware is classified according to its architecture in order of increasing computing power:

- Single core processors: this architecture was the workhorse of space application until the recent years and it is still considered for less demanding functionalities because of its long heritage reliability. The ERC32 and the LEON processors are probably the most representative and used for space applications. Also the ARM architecture provides some notable processors. Some models of the ARM architecture offers some specific support for AI/ML algorithms (e.g. some ISA extension suited for quantization).
- Multicore processors: this architecture is the natural evolution of the single-core architecture: performance are improved by adding more cores. The CAES GR740 ([15]), based on the updated LEON architecture (LEON4 – SPARC V8) is the present state of the art and is the basis for missions currently under development. For the next future, the RISC-V architecture will probably catch on. RISC-V is more performant and license-free; it also offers an expandable ISA that offer better support to AI/ML algorithms and quantization.
- Manycore processor: this architecture is based on a set of heterogeneous processors (that can also include HW accelerators) that are able to efficiently manage a wide set of problems. Manycore processors are generally based on general purpose processors and DSPs or hardware accelerators (e.g. GPUs). This architecture can be considered a further evolution (and the successor) of the multicore architecture.
- GPUs and VPUs: GPUs rapidly manipulate and alter memory to accelerate the creation of images for output to a display device. Their highly parallel structure makes them more efficient than general-purpose CPUs. VPUs represent a new class of processors able to increase the speed of visual processing as CNN, SIFT and similar. They are targeted toward mobile applications and are optimized for small size and power efficiency. These architectures are not stand-alone and generally integrated in more complex hardware design.
- TPU (Tensor processing Unit): this architecture is based on AI accelerator ASIC specifically designed for neural network machine learning. They are generally used form NN training into clouds infrastructures.
- Neuromorphic processor: very-large-scale integration (VLSI) systems containing electronic analog circuits to mimic neuro-biological architectures present in the nervous system. A neuromorphic computer/chip is any device that uses physical artificial neurons (made from silicon) to do computations).
- FPGA and SoC: Field Programmable Gate Array is an integrated circuit that can be configured using a Hardware Description Language. In the recent years the FPGA are included into a SoC, i.e. a device including, beyond FPGA, multiple fabric CPU cores, allowing for increased flexibility when designing an application. They are very flexible devices that allow to implement domain specific architectures. They allow to easily reconfigure an hardware design and they generally offer the best trade-off between performance and costs.
- ASIC: An application-specific integrated circuit is an integrated circuit (IC) chip customized for a particular use, rather than intended for general-purpose use. They generally provide the best

performances and power efficiency but they are suited for very high-volume mass production and they provide no flexibility to changes.

Benchmarks and comparative analysis about the state of the art hardware can be found in [16], [17], [18], [19], [20]. Surveys and more general information can be found in [21], [22], [23], [24].

Some of the architectures listed above cannot be considered for space missions at the moment: TPUs and Neuromorphic processors suffer of lack of flexibility and space grade solutions; ASICs are very expensive and not reprogrammable .



As regards LEO and short term missions context, a lot of studies and analysis, based on the use of last generation COTS HW solutions, are already underway ([25], [26], [27], [28]). These studies include the use of the most performing architectures (i.e. many-core, GPUs/VPUs, FPGAs/SoCs) that offer various degrees of parallelism and extended instruction set that is able to speed up the AI/ML operations. These architectures fully support the use of smaller precision numbers (i.e. quantization) to reduce the memory footprint by providing an hardware that is able efficiently use these type of values. FPGA based solutions allow the development of hardware fully customized in the domain problem and the most evolved models (e.g. Xilinx Versal) promise a quantum leap in terms of hardware performance. However, the greater application memory footprint of FPGA solutions could be a strong disadvantage in case of frequent updates and reduced bandwidth.

In the context of the most challenging environmental conditions (LEO, long-lasting, deep space) the introduction of more performing hardware is much slower and many of the solutions listed above cannot be used. For these type of missions, the multicore CAES GR740 is the state of the art and the reference point in the short and medium term; this hardware more than quadruples the performances of the previous generation of RAD Hard processors but its performances are not comparable with more performing architectures (manycore, GPUs/VPUs, FPGAs/SOCs, etc.). Since a multicore processor is typically used to group multiple functions in an unique board, the computational power has to be shared between various applications. In the context of GEO missions it is not rare to see even less performing solutions such as the CAES UT32M0R500 (based on the Cortex-M0+ architecture). This precludes the possibility to run the most complex AI/ML algorithms. The architecture of these processors are not meant to deal with typical AI/ML computations. These processors have not specific ISA support for smaller precision numbers, thus using quantization can reduce memory footprint but it has no effect (or even a negative effect) on algorithm performances.

3. Spacecraft Dataset

A use-case anomaly scenario is proposed hereafter in order to test and benchmark the effectiveness of the proposed algorithm solution. The dataset contains telemetry data related to the on-board Attitude Control Subsystem, more specifically to the Reaction Wheels (RWs) actuators. This kind of actuators are generally composed by an external part fixed to the spacecraft, and an internal rotating flywheel which is driven by an electric motor, and they are typically selected in configurations of three or more units for redundancy; they are generally used to apply a continuous fine attitude control over the orbit during the nominal spacecraft operation, or to react to the environmental disturbance torque. The considered telemetry data is the absorbed current of the Reaction Wheel, which is directly correlated with its angular speed and with the

friction experienced by the rotating flywheel. The anomaly is simulated by faking a dummy sudden increase of internal flywheel friction, which in turns suddenly increases the wheel's absorbed current, as shown in Figure 1.

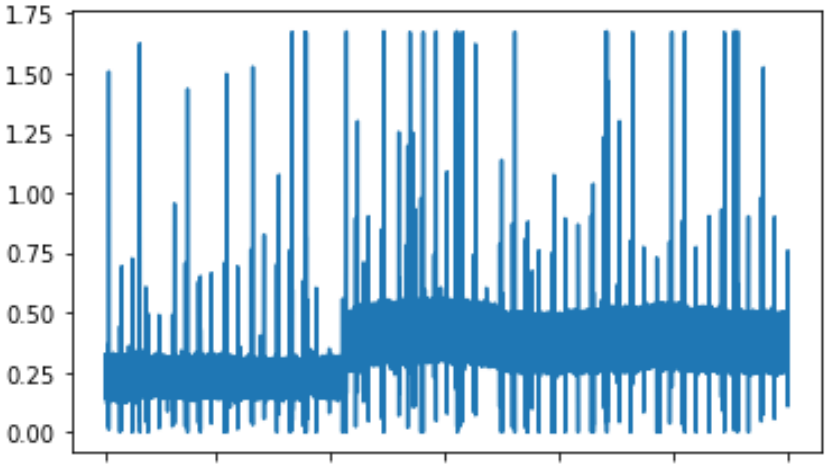


Figure 1: Telemetry sensor readings containing anomalous sample

4. AutoEncoder for on-board Anomaly Detection

As anticipated in Section 1.1, an AutoEncoder-based detector compress some meaningful information extracted by non-linear projection of the input observation $x_i \in R_n$, into a manifold living in a lower dimensional space $z_m \in R_m$, with $m < n$. This is also the input of a decoder stage which produces as output a vector $\hat{x} \in R_n$ designed to be a replication of x (see Figure 2).

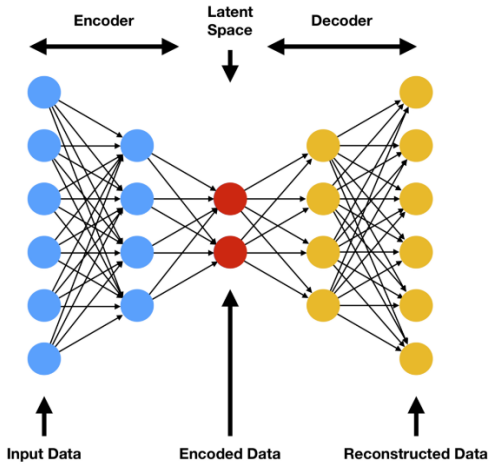


Figure 2: AutoEncoder basic structure

The input size of the encoder $x_i \in R_n$ is equal to the dimension of the time series samples given as input to the model, here defined as window of fixed dimension n, which are non-overlapping each other. It has been proven that there is a correlation between the increasing number of observed samples and the performance of the model itself [5]; however, with increasing input dimension there is also an increasing memory occupancy of the resulting neural network (and associated computations), because of the input neurons. The dimension of these sliding window, which is a hyper-parameter, has been decided to set to $n = 300$, which are equivalent to 10 minutes of collected sensor data.

The definition of the window dimension hyper-parameter is preparatory for the pre-processing stage (that can be visualized in Figure 3), where the dataset, both training and test set, is accordingly modified as such:

i) data polishing of not nominal data
ii) $minmax$ normalization are performed.

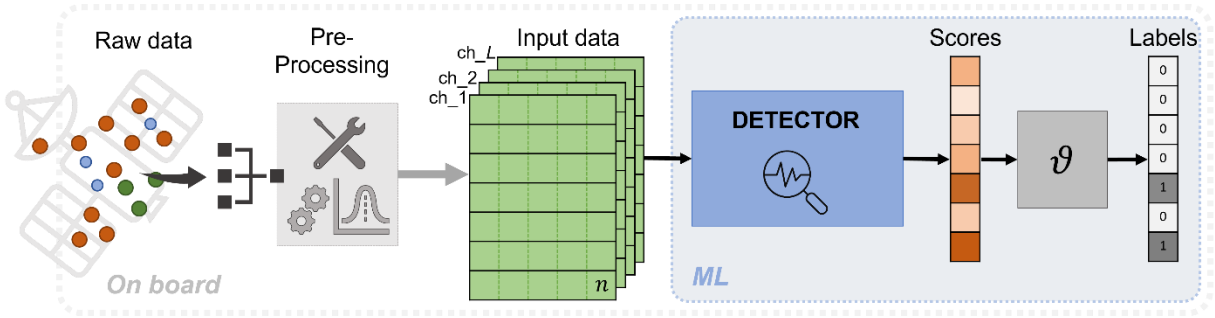


Figure 3: Machine Learning Pipeline for telemetry anomaly detection

The data polishing is a required step in the pipeline, because the dataset at our disposal contains samples associated to non-nominal modality of the satellite, like a manoeuvre or a pointing for recharging the batteries, that shall not identified as anomalies. We analysed the cardinality of these values and noticed that the ratio between the number of nominal samples and not-nominal is exiguous, specifically:

$$Ratio(nominal, non-nominal) < 3e-4\%$$

Therefore, considering the limited occurrences of these sample, it has been decided to remove these values from the training and test set, without generating synthetized ones that can erroneously introduce anomalous sample.

Finally, we normalized all the values of the dataset applying the min-max normalization, also known as min-max scaling, which consists in rescaling the range of values to scale the range in $[0,1]$ or $[-1,1]$. The formula for the implemented min-max scaling of $[0,1]$ is given as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \qquad (1)$$

where x is the original value, and x' it the normalized value.

The basic AutoEncoder structure is modified in order to face the limitation of spacecraft on board computer, as described in the next sections.

4.1 AutoEncoder structure

The AutoEncoder inner structure is subdivided in an Encoder and Decoder model:

- the Encoder is implemented as fully connected layers concatenated that implement a nonlinear function such that
$$f(x_i): R_n \rightarrow R_m, m < n,$$
- the Decoder is the transposed structure of the Encoder and implement a nonlinear function such that:
$$g(z_i): R_m \rightarrow R_n, m < n$$

This architecture is detailed in Table 1.

Table 1: AutoEncoder structure

	Layer	Units	Non Linearity
Encoder	Linear 1	300	ReLU
	Linear 2	150	ReLU
	Linear 3	50	ReLU
	Linear 4	25	ReLU
Decoder	Linear 5	25	ReLU
	Linear 6	50	ReLU
	Linear 7	150	ReLU
	Linear 8	300	ReLU

We trained the AutoEncoder using a semi-supervised learning approach, namely only nominal samples (or samples associated to a single class) are given as input during training phase: doing so the model *learn* the latent space of nominal dataset, therefore it is capable of reconstruct nominal data.
Moreover the network parameters are trained by minimizing a loss function given by the Mean Squared Error

We can leverage this capabilities of input reconstruction by defining what is called reconstruction error: the drift between the input value x_i and the output of the model $\hat{x}_i = g(f(x_i))$ defines in a quantitative way how much the input deviates from the learned nominal subspace. Higher the value, higher is the probability that the input is anomalous.
We mathematically define the metric anomaly score, as:

$$AnomalyScore = MSE\left(x_i - g\left(f(x_i)\right)\right) \qquad (2)$$

This is however a relative and not-absolute metric, therefore, we have to define a decision criteria that indicates where the analysed window samples are anomalous or not. We implemented this criteria leveraging the nature of the anomaly score computed on nominal samples, which will be as low as possible. In fact, after the learning phase, we performed an inference on the same training set computing the anomaly score. From this results we mathematically define a threshold value, which is an upper bound to the anomaly score of the input sample, specifically:

$$Threshold = max(AnomalyScore(training\_set)) + \sigma \qquad (3)$$

Where σ is one standard deviation of the mean computed on the nominal training set. Therefore a sample is identified as anomalous if its anomaly score exceed this value.

4.2 Quantization-Aware Training (QAT)

Quantization is the process of reducing the precision of the weights and activations of a neural network model in order to make it more efficient to run on hardware with limited computational resources. A model can be quantized by converting it to an "int8" representation, which reduces the precision of the weights and activations from 32-bit floating point numbers (FP32) to 8-bit integers (INT8).
In this work we focus on Quantization Aware Training, where all weights and activation are fake quantized during both forward and backward passes of training and the quantization is simulated during training: that is, 32-bit float values are rounded to mimic int8 values, but all computation are still done with floating point numbers. Thus, all the weight adjustments during training are made while "aware" of the fact that the model will ultimately be quantized; after quantizing, therefore, this method usually yields higher accuracy than PTQ method [11] for small models.

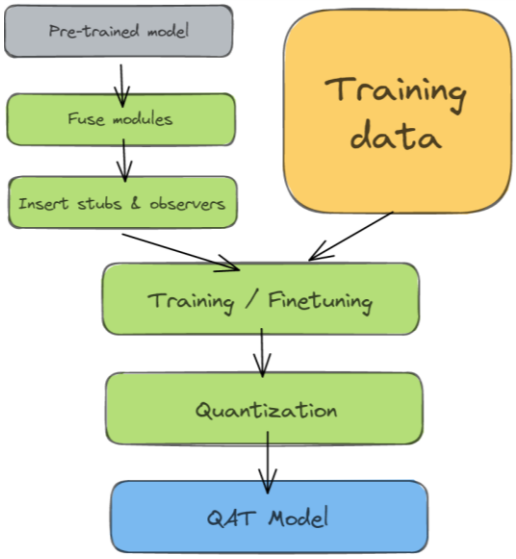


Figure 4: Quantization-Aware Training step[1]

These are the required steps for the QAT of the AutoEncoder:

1. *Model Fusion*: this step combines multiple sequential modules into one. Fusing means that the compiler need to only run one kernel instead of many, speeding up the inference phases. This step needs to be done manually depending on the model architecture;
2. *Stub insertion*: the classic AutoEncoder module is not ready for QAT, that's because it accepts floating point values as input. To get around this problem we shall insert the Quantization Stub, which converts input tensors from floating point to quantized;
3. *Training and convert*: finally run the training process and convert the observed model to a quantized one. During training, all calculations are done in floating point, with fake quantization tensors modelling the effects of quantization by clamping and rounding to simulate the effects of INT8. After model conversion, weights and activations are really quantized, and activations are fused into the preceding layer where possible.

4.2.1 QAT implementation problems and limitation

To perform the above mentioned procedure 2 requires a Quantized Engine to be specified and used for execution. It is important the engine used for quantized computations are the same of the backend on which the model will be deployed and executed.

Up to now the most diffused commercial framework for Neural Network implementation offers a very limited set of backend engines, like FBGEMM 3 for quantized operators on x86 machines or QNNpack 4 for ARM CPUs. This proves however a limitation for the space qualified hardware, which is not based on ARM or x86 CPUs, as described in Section 2.22. This limitation can be overcome implementing a custom backend for different CPU architectures or implementing the models using a different low-level language like C.

[1] https://pytorch.org/blog/quantization-in-practice/#quantization-aware-training-qat
[2] https://pytorch.org/docs/stable/quantization.html#quantization-flow
[3] https://github.com/pytorch/FBGEMM
[4] https://github.com/pytorch/QNNPACK

The implementation of this method, however, exceeds the boundaries of this study, as the focus is on determining the feasibility and applicability of this approach within a limited hardware performance environment. We leave these implementation as continuation of the study.[5]

4.3 Pruning

There are several ways to perform pruning, including weight pruning, filter pruning, and neuron pruning. In weight pruning, individual weights in the network are set to zero if they are below a certain threshold. Filter pruning removes entire filters in a convolutional layer, while neuron pruning removes entire neurons in a fully connected layer.
In addition to reducing the size of the network, pruning can also improve its generalization ability and make it more robust to adversarial examples. This is because pruning helps to eliminate redundant and over-fitting neurons, which can be especially useful in situations where the training data is limited.

We implemented a weight pruning of the model, removing a specified amount of units with the lowest L1-norm, in order to reduce the model complexity and computations as we can see in Section 5.

5. Results

The discussed AutoEncoder was trained and tested in three different configurations for comparison purpose, where both training and test set are passed to a pre-processing stage where filtering and data standardization are performed. More in details, we implemented three different AutoEncoder optimization configurations:

- non optimized model
- quantized model, using INT8 quantization
- model weight pruning to which is applied quantization

5.1 Anomaly identification

Detectors' accuracy is assessed by showing the anomaly score computed by all the three abovementioned configurations of the AutoEncoder, of both period containing a step anomaly (test set) and only nominal sample (training set). Moreover, we depict also the histograms of the anomaly score values for each method and for all mentioned cases.

In order to identify the anomalies on the test set, we have to define the anomaly threshold as described in Section 4.1. The computed thresholds and associated values (i.e. mean, max and standard deviation) are grouped in Table 2.

Table 2: AutoEncoder identified threshold from nominal training set

	Mean	Max	Std	Anomaly threshold
AE base	0,0143	0,0257	0,0257	0,0283
AE quantized	0,0161	0,0311	0,0030	0,0342
AE pruned and quantized	0,0191	0,0349	0,0036	0,0385

We can leverage these values on the computed anomaly score using the three model configurations.
We can see in Figure 5 that the base AutoEncoder is capable of identifying the step anomaly, corresponding to the increase in the computed anomaly score.

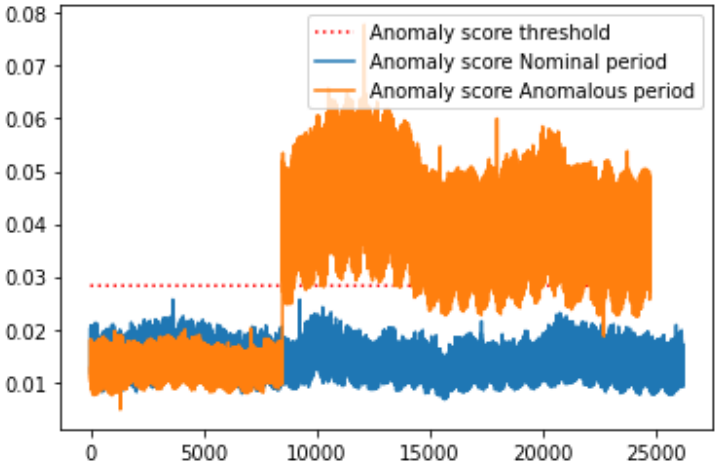


Figure 5: Non Quantized model anomaly score

Comparing Figure 5 and 6 we can see that the anomaly score of the quantized model does not drift significantly from the base model, as expected and anticipated in Section 1. The quantized model has similar performance to the base model, where the anomaly is correctly identified. The main differences can be seen in the value of the anomaly score associated to the anomalous period and the identified threshold from nominal one. In fact the anomaly threshold has an higher base value, meaning that the base model is more capable to reconstruct the nominal input; this reflect also on the score of the anomalous period where the mean value is higher than the base case.
Moreover in Figure 7 we show the plot of the anomaly score of the pruned and then quantized model. Also in this case the model results does not differ significantly, obtaining a similar trend in anomaly score, correctly identifying the anomaly.

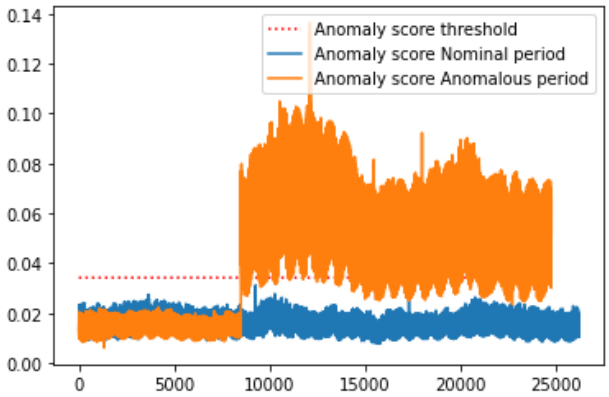


Figure 6: Quantized model anomaly score

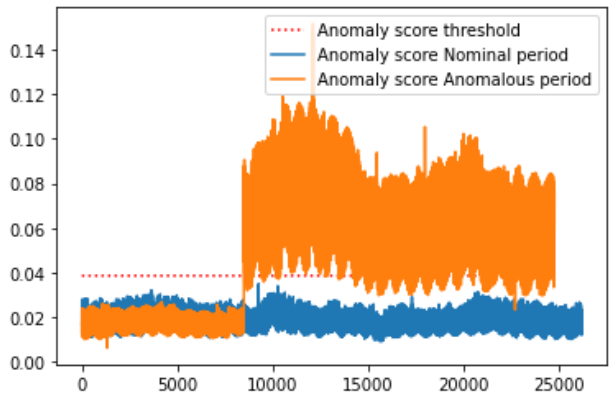


Figure 7: Pruned and Quantized anomaly score

Fig. 8, 9, 10 depicts the histograms of the anomaly scores in all the three configurations. We can notice that on all the configurations two Gaussians can be seen. This behaviour follows the anomaly shown in figure 5, 6, 7 in which there is a sudden increase in values. Therefore the distribution with lower mean is associated to the first portion of telemetry data not containing an anomaly, while the distribution with higher mean (and so higher anomaly score) is associated to the portion of telemetry data containing an anomaly.

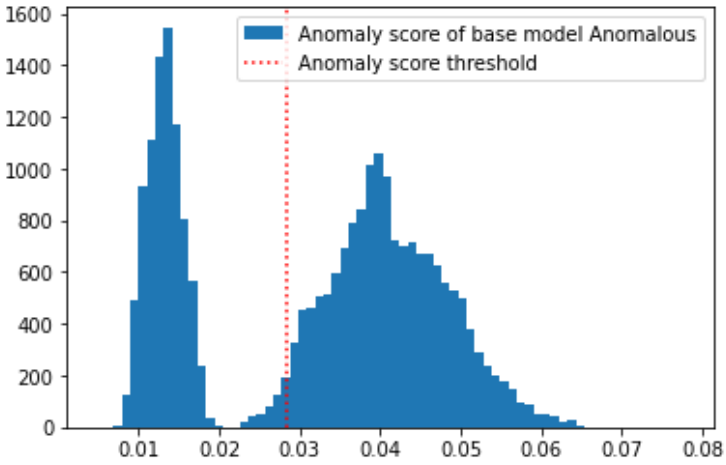


Figure 8: Histogram of anomaly score of base model (anomalous period)

We can notice that the anomaly distribution of the base model (Figure 8) has lower variance with respect to the quantized and pruned models (Figure 9, 10): therefore the base model is capable of associate similar high anomaly scores to the anomalous samples.

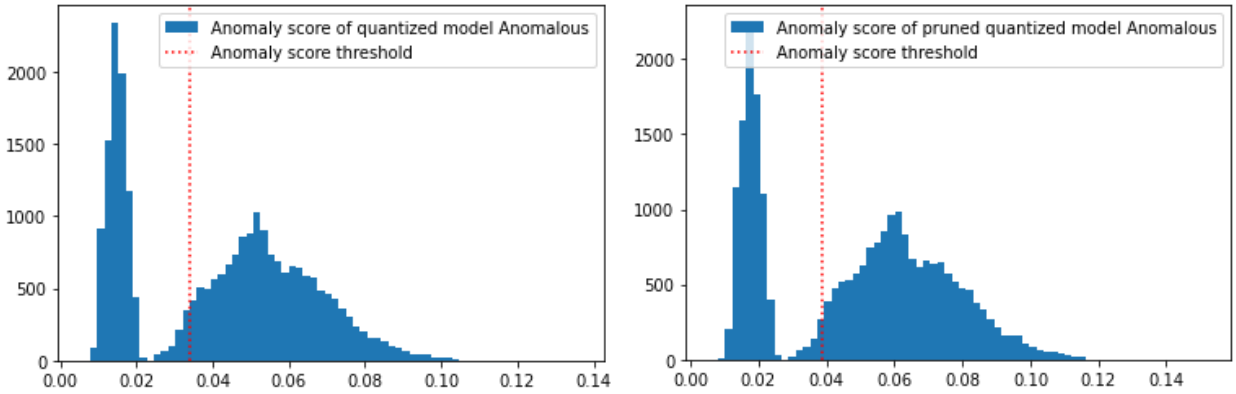


Figure 9: Histogram of anomaly score of quantized model (anomalous period)

Figure 10: Histogram of anomaly score of pruned and quantized model (anomalous period)

5.2 Memory occupancy reduction and MADDs

As already explained the quantized models has as major outcome the decreasing the memory occupancy. We can notice this decreasing in model weight in the Table 3.

Table 3: AutoEncoder memory occupancy in different configurations

Base model (Mb)	Quantized model (Mb)	Pruned and Quantized model (Mb)
1.502	0.136	0.136

The quantized model has a significant weight reduction in Mb, more specifically the quantized model weigh only 9.05% of the base original model, even obtaining similar results in terms of anomaly identification. With no surprise the pruned model has the same weight in terms of Mb, this because the weight pruning of the model does not affect the Mb weight.

Moreover, recent works shifts the focus from reducing parameters to reducing the number of operations (MADDs – Multiply and Add operations). To reduce this metric, we applied weight pruning to the model, obtaining a reduction in terms of MADDs of about 14.2% with respect to the base model.

Table 4: AutoEncoder MADDs in different configurations

Base model (M)	Quantized model (M)	Pruned and Quantized model (M)
67.200	67.200	58.368

More pruning methods can be implemented, like bias pruning, in order to obtain a reduction in the MADDs metric. However in this work we focused only on the weight pruning, leaving these possibility to the follow-up of this study.

6 . Conclusions

In this work, we have described the work-in-progress about the investigation and application of feature extraction and anomaly detection techniques for analysing spacecraft telemetries typically generated from LEO satellites. We assessed the performance of a well-known ML method, the AutoEncoder, properly tuned for the task of anomaly detection for on-board spacecraft health monitoring systems. A key point in our research is the building up of the model optimization with different configurations, like quantization (from floating point to integer precision) for memory reduction and weight pruning for MADDs reduction. We obtained significant memory occupancy reduction and a reduction in the MADDs metric, without losing significant performance in anomaly identification.

Recorded performance results suggest that the ML models are worth deploying for the application of interest.

In the roadmap of our research, further investigations and developments are planned, e.g. in terms of validation of the model with new known anomalies, model implementation on space qualified hardware (by direct programming in C, or implementing an ad hoc backend engine), implementation of different pruning methods of the network, simultaneous monitoring of several telemetry quantities in the same window (multivariate time series analysis), scalability and reliability analysis of developed ML models.

References

[1] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola e R. Williamson, «Estimating Support of a High-Dimensional Distribution,» *Neural Computation*, vol. 13, pp. 1443-1471, 2000.

[2] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. Mahoney e K. Keutzer, «A Survey of Quantization Methods for Efficient Neural Network Inference,» *arXiv*, 2021.

[3] D. Ballard, «Modular learning in neural networks,» *Proceedings AAAI*, 1987.

[4] X. Bampoula, G. Siaterlis, N. Nikolakis e K. Alexopoulos, «A Deep Learning Model for Predictive Maintenance in Cyber-Physical Production Systems Using LSTM Autoencoders,» *Sensors*, 2021.

[5] C. Ciancarelli, F. Corallo, S. Cognetta, E. Mariotti, L. Manovi, M. Mangia, A. Marchioni, R. Rovatti, F. Pareschi e G. Setti, «New Concepts Of Automated Anomaly Detection In Space Operations Through ML-Based Techniques,» *73rd International Astronautical Congress (IAC)*, 2022.

[6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adame D. Kalenichenko, «Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,» *arXiv*, 2017..

[7] K. He, X. Zhang, S. Ren e J. Sun, «Deep Residual Learning for Image Recognition,» *arXiv*, 2015.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens e Z. Wojna, «Rethinking the Inception Architecture for Computer Vision,» *arXiv*, 2015.

[9] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney e K. Keutzer, «A Survey of Quantization Methods for Efficient Neural Network Inference,» *arXiv*, 2021.

[10] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen e T. Blankevoort, «A White Paper on Neural Network Quantization,» *arXiv*, 2021.

[11] H. Wu, P. Judd, X. Zhang, M. Isaev e P. Micikevicius, «Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation,» *arXiv*, 2020.

[12] "TensorFlow framework," [Online]. Available: https://www.tensorflow.org/. [Accessed 16 05 2022].

[13] "PyTorch framework," [Online]. Available: https://pytorch.org/. [Accessed 16 05 2022].

[14] D. d. D. Benoît, "A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor," in *International Conference on Computational Science*, 2013.

[15] «GR740 Next Generation Microprocessor Flight Models,» in *TEC-ED & TEC-SW Final Presentation Day*, 2021.

[16] M. Ghiglione, A. Raoofy, G. Dax, G. Furano, R. Wiest, C. Trinitis, M. Werner, M. Schulz and M. Langer, "Machine Learning Application Benchmark for in-orbit on-board," in *European Workshop on-board data processing*, 2021.

[17] T. M. Lovelly, *Comparative analysis of space-grade processors*, University of Florida, 2018.

[18] M. L. Tyler e A. D. George, «Comparative analysis of present and future space-grade processors with device metrics,» *Journal of Aerospace Information Systems*, 2018.

[19] G. Lentaris, K. Maragos, I. Stratakos, P. Lazaros, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzales-Arjiona and G. Furano, "High-Performance Embedded computing in space: Evaluation of Platforms for Vision-Based Navigation," *Journal fo Aerospace informatin systems*, 2018.

[20] T. M. Lovelly, T. W. Wise, S. H. Holtzman e A. D. George, «Benchmarking Analysis of Space-Grade Central Processing Units and Field-Programmable Gate Arrays,» *Journal of Aerospace Informatin Systems*, 2018.

[21] A. Reuther, P. Michaleas, J. Michael, G. Vijay, S. Siddharth e K. Jeremy, «Survey of Machine Learning Accelerators,» in *IEEE-HPEC conference*, Waltham, 2020.

[22] A. D. George e C. Wilson, «Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites,» *Proceedings of the IEEE*, vol. 106, pp. 458-470, 2018.

[23] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang e Others, «Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?,» *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 5-14, 2017.

[24] S. Di Mascio, A. Menicucci, E. Gill, G. Furano and C. Monteleone, "Leveraging the openess and modularity of RISC-V in space," *Journal of Aerospace Information Systems*, vol. 16, no. 11, 2019.

[25] B. Garabedian, «AMD Announces Completion of Class B Qualification for First Space-Grade Versal Adaptive SoCs Enabling On-Board AI Processing in Space,» 15 11 2022. [Online]. Available: https://www.amd.com/en/press-releases/2022-11-15-amd-announces-completion-class-b-qualification-for-first-space-grade. [Consultato il giorno 25 01 2023].

[26] S. Fratini, J. Gorfer e N. Policella, «On Board Autonomy Operations for OPS-SAT Experiment,» *Appl Intell*, n. 52, 2022.

[27] G. Giuffrida e e. al., «The Φ-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-14, 2022.

[28] «State of the art of Smallsat Space Avionics,» NASA, Ames Research Center, Moffett Field, California, 2021.

[29] G. Giuffrida et al., «The Φ-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-14, 2022.

[30] P. Zetocha, «Comparison of AI technologies for satellite anomaly FDIR,» *37th Aerospace Sciences Meeting and Exhibit*, 1999.