scientific reports

Check for updates

OPEN

Efficient adaptation of deep neural networks for semantic segmentation in space applications

Leonardo Olivi[1], Edoardo Santero Mormile[2] & Enzo Tartaglione[3]✉

In recent years, the application of Deep Learning techniques has shown remarkable success in various computer vision tasks, paving the way for their deployment in extraterrestrial exploration. Transfer learning has emerged as a powerful strategy for addressing the scarcity of labeled data in these novel environments. This paper represents one of the first efforts in evaluating the feasibility of employing adapters toward efficient transfer learning for rock segmentation in extraterrestrial landscapes, mainly focusing on lunar and martian terrains. Our work suggests that the use of adapters, strategically integrated into a pre-trained backbone model, can be successful in reducing both bandwidth and memory requirements for the target extraterrestrial device. In this study, we considered two memory-saving strategies: layer fusion (to reduce to zero the inference overhead) and an "adapter ranking" (to also reduce the transmission cost). Finally, we evaluate these results in terms of task performance, memory, and computation on embedded devices, evidencing trade-offs that open the road to more research in the field. The code will be open-sourced upon acceptance of the article.

The exploration of celestial bodies beyond Earth requires deploying sophisticated computer vision systems endowed with the capability to robustly identify and characterize surface features[1]. Among the critical tasks in this extraterrestrial context, rock segmentation plays a critical role, since the geological composition of these distant terrains offers vital insight into the history and potential habitability of celestial bodies[2,3]. However, the acquisition of labeled data for training machine learning models in such environments proves to be an arduous challenge, primarily due to the impracticality of physical presence and the associated complexities of data collection.

Another obstacle lies in the difficulty of transmitting large volumes of data from remote celestial bodies back to Earth[2]. The limited bandwidth and communication capabilities of space missions of impose constraints on the efficiency and feasibility of transmitting extensive datasets. Consequently, this limitation hinders the traditional approach of gathering abundant labeled data for training and updating machine learning models deployed in extraterrestrial environments.

Besides, the dynamic nature of extraterrestrial landscapes, subject to geological and environmental variations, underscores the need for adaptive and updatable models. Continuous updates to machine learning models become imperative to account for changes in surface features, ensuring that the algorithms remain effective and accurate over extended mission durations. Efficient transmission of these updates becomes a critical consideration as it involves optimizing the use of limited bandwidth to relay pertinent model adjustments without compromising the overall mission (see Fig. 1).

In response to the formidable challenge of acquiring labeled data for training machine learning models in extraterrestrial environments, we present a pioneering exploratory study leveraging the concept of *adapters* applied to a pre-trained backbone[4] for space applications and efficient memory-saving strategies. Concretely, we contribute in three different aspects.

- Definition of the structure of adapters properly suited for this task (Section "Adapters design"). Although many designs have been proposed in recent years, we customize the design to be adaptable to the realistically deployable architectures, reducing transmission and inference costs.
- Analyze the real memory complexity for the inserted modules (Section "Memory complexity of adapters"). This aspect covers a marginal role in other works where adapters are proposed, but in our design, the memory footprint covers a central role in the solution's deployability.
- Implementation of two strategies that enable the deployability of the proposed method: *adapter fusion*, where we explain how to fuse the adapters to the original backbone, causing no computational overhead (Section

[1]Turin, Italy. [2]University of Trento, Trento, Italy. [3]LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France. ✉email: enzo.tartaglione@telecom-paris.fr

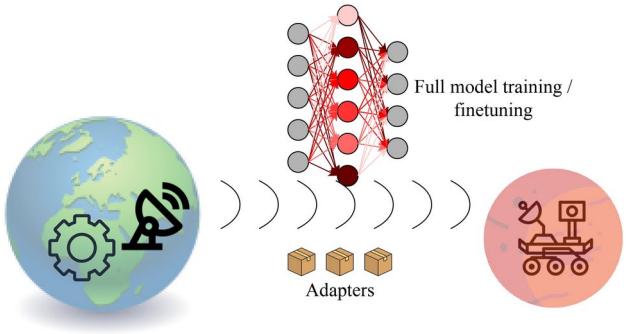


Fig. 1. Transmitting a fully fine-tuned model can be very bandwidth and memory-demanding. In this work, we explore the possibility of employing adapters, shallow modules that introduce corrections to a pre-loaded one (and that can be considered as an update).

"Adapter fusion"), and *adapter ranking*, allowing us to selectively choose which layers require adaptation, saving bandwidth for the update (Section "Adapters ranking").

We validate our study through a quantitative evaluation of the task of interest, evaluating trade-offs in terms of the size of the VS model in performance and the computation required, showing that adapters provide a fair compromise between performance and memory size (Section "Experiments"). This study is also completed by measures on real embedded devices, either GPU equipped or not.

Related work

Traditional approaches to rock segmentation

The task of rock segmentation in extraterrestrial environments deepens its roots far back to the past. The work by Gor et al. provided one of the first possible setups that allowed autonomous rock segmentation for Mars rovers[2]. Specifically, in this work, the authors introduce a parameter-independent framework for autonomous rock detection in Martian terrain, with a demonstrated algorithm applied to real Mars Rover data. The idea behind this work was to build foundations for all the next coming work: facilitate efficient decision making, aid in prioritizing data for transmission, select regions for in-depth scientific measurements, and optimize exploration strategies to maximize scientific returns per transmitted data bit. This was achieved through clustering algorithms. Thompson and Castrono performed a first comparison between seven segmentation algorithms (including Support Vector Machines) was performed by Thompson and Castaño[5], where the best overall performance was attributed to the *Rockfinder* algorithm[6] having target selection accuracy rates[6] of approximately the 80%, but highlighting as well that finding all rocks in the image is a difficult task, obtaining a 60% recall. Other approaches in later years tried to improve performance on this task, including the use of random forests to perform semantic segmentation for hazard detection on planetary rovers[7], edge regrouping[8], the exploitation of region contrast to aid in segmentation[9], superpixel graph cuts[10] and including size-frequency priors in the fitting analysis[11].

Deep learning for rock segmentation

Although traditional approaches are a better fit for working in aerospace environments where computation and memory are very limited, the performance of Deep Learning (DL) approaches shows that moving towards these new techniques is currently the most promising direction despite the high computational costs[3]. The first pioneering approach tried to overcome low generalization performance with the employment of Support Vector Machines[12]: since traditional visual segmentation techniques struggle with the diverse morphologies of rocks, this work integrates features from multiple scales. More recently, several recent advancements in rock segmentation for extraterrestrial environments have been proposed, showcasing diverse approaches to address the challenges of navigation and classification in extraterrestrial environments. Kuang et al.[13] employ a variant of U-Net++[14] to achieve rock segmentation for Mars navigation systems. Deep Mars[15] trains an AlexNet to classify engineering-focused rover images, albeit limited to recognizing a single object per image. The Soil Property and Object Classification[16] introduces a fully convolutional neural network for Mars terrain segmentation. Swan et al.[17] contribute a terrain segmentation dataset and assess performance using DeepLabv3+[18]. Notably, Transformer-based networks have emerged as a focus, with studies such as RockFormer[19] and MarsFormer[20] demonstrating their efficacy in Martian rock segmentation. Addressing broader visual navigation challenges, Zhang et al. [21] tackles Mars' visual navigation problem through a deep neural network capable of finding optimal paths in the global Martian environment. Finally, the space hardware limitations, spanning across ensuring their memory and computational efficiency, constrain heavily the DL implementations, requiring a careful and deepened study of the optimization of these, Marek et al. [22].

Transfer learning approaches for deep learning

In the context of future extraterrestrial environments, the pivotal role of DL is anticipated; however, its effectiveness is impeded by the inherent challenge of acquiring annotated training data. Traditional machine learning algorithms often presuppose that training and testing data must inhabit the same feature space and exhibit similar distributions[23], a

condition that proves challenging in the context of obtaining an adequate quantity of Martian rock images. Addressing this, knowledge transfer emerges as a promising avenue to enhance learning performance and alleviate the challenges associated with dataset acquisition in extraterrestrial scenarios. Li et al. [24] categorize transfer learning approaches into four distinct scenarios. Firstly, *parameter-transfer* involves identifying shared parameters or prior distributions between the source and target domain models, aiming to enhance learning performance[25]. Secondly, *feature-representation-transfer* focuses on learning representative features through the source domain, with the encoded knowledge transferred across domains embedded within the learned feature representation[26,27]. The *relational-knowledge-transfer* scenario assumes relational and independently identically distributed source and target domains, aiming to establish a mapping of relational knowledge between the two[28]. Lastly, *instance-based transfer* involves the reuse of a portion of data from the source domain for learning in the target domain through a process of reweighting[29,30]. These categorizations provide a comprehensive overview of the diverse strategies employed in transfer learning scenarios. In our work we will be falling into the parameter-transfer scenario, adapting the parameter's distribution efficiently with the employment of a solution inspired by adapters[4]. Such an approach also shows further potential to reduce the model's complexity, employing pruning at the adapter's scale[31].

Challenges of deploying neural networks in space environments

Despite such a high level of performance, deploying neural network-based models for image segmentation in space exploration presents unique challenges due to the constrained operational environment[32]. Space-borne systems operate with significantly limited computational resources compared to terrestrial applications, as space-qualified hardware has lower processing power and memory[33,34], necessitating model optimization techniques such as quantization, pruning, and adapter-based fine-tuning[35–38]. Communication constraints further complicate deployment, as deep space missions suffer from high-latency and low-bandwidth transmission, making real-time updates impractical. Autonomous models must be capable of in-situ learning and quick adaptation to maintain performance under changing environmental conditions or rapidly evolving environments[39]. Furthermore, due to such a quick and active environment, these models need to be robust to noise and malfunctions, necessitating fault-tolerant AI implementations. Understanding and quantifying the impact of such issues on DL models deployed remains under-explored[40], but it is a challenge to address in this field. Solutions like providing a custom design relying on annotated images[41]. These constraints underscore the necessity of designing neural networks that are efficient and adaptable, particularly for image segmentation tasks in space exploration scenarios.

Efficient adaptation of a DL model

In this section, we will provide an overview of the techniques chosen to make a Deep Neural Network (DNN) able to work efficiently in different environments, namely by using adapters. We motivate their choice for our specific setup, their architecture, and implementation, together with our *adapter fusion* and *adapter ranking*.

Adapters design

The main inspiration for the adapters injection comes from the work by Rebuffi et al. work[42] concerning residual adapters originally introduced in a ResNet-15 model. Within the cited works, the design of residual adapters has shown great performance, outperforming standard transfer learning techniques. Let y_n be the output of the n-th layer in a DNN model. We can generally model layers in recent popular models like ResNets and even more traditional ones like VGGs can be expressed as

$$y_n = \varphi_n \left\{ \mathrm{BN}_n \left[f_n \left(x_n, w_{f_n} \right), w_{\mathrm{BN}_n} \right] \right\}, \tag{1}$$

where φ_n is the applied non-linearity, BN_n is the Batch normalization layer, f_n is the linear transformation applied (*e.g.* fully connected or convolutional), and w_{f_n} and w_{BN_n} are the learnable parameters associated to f_n and BN_n, respectively.

Let us assume our model is trained on some upstream task T_0 and should be fine-tuned on a downstream task T_1: in the case we update the parameters of the n-th layer, we have

$$y_n^{T_1} = \varphi_n \left\{ \mathrm{BN}_n \left[f_n \left(x_n, w_x^{T_1} \right), w_{\mathrm{BN}_n}^{T_1} \right] \right\} \tag{2}$$

having

$$w_x^{T_1} = w_x^{T_0} + \Delta w_x^{T_1}, \tag{3}$$

where $w_x^{T_0}$ indicates the original parameters from T_0 and $\Delta w_x^{T_1}$ is the parameter's change after fine-tuning on T_1. If T_1 is very correlated with T_0, we can expect that $\Delta w_x^{T_1} \approx 0$, meaning that its dimensionality can be reduced. Hence, instead of learning $w_x^{T_1}$ initializing it to $w_x^{T_0}$, we can learn $\Delta w_x^{T_1}$ directly: we can inject then, between f_n and BN_n, a *series adapter*

$$A_n^{T_1} = f_{A_n} \left[f_n \left(x_n, w_{f_n}^{T_0} \right), w_{f_{A_n}}^{T_1} \right] \tag{4}$$

entitled to learn the parameter's shift for T_1 implicitly and to be directly plugged into the DNN. At this point, (2) writes

$$y_n^{T_1} = \varphi_n \left\{ \mathrm{BN}_n \left[A_n^{T_1} \left(f_n \left(x_n, w_x^{T_0} \right), w_{f_{A_n}}^{T_1} \right) + f_n \left(x_n, w_x^{T_0} \right), w_{\mathrm{BN}_n}^{T_0} \right] \right\}. \tag{5}$$

In our design, we want f_{A_n} to maintain the same output dimensionality; hence good choices for it are 1×1 convolutions or Batch Normalizations; we choose for our design to employ both:

$$A_n = f_{A_n}\left\{\mathrm{BN}_{A_n}\left[f_n\left(x_n, w_{f_n}^{T_0}\right), w_{\mathrm{BN}_{A_n}}\right], w_{f_{A_n}}\right\}. \tag{6}$$

A visual representation of (6) can be found in Fig. 2.

Memory complexity of adapters

Let us analyze the complexity of employing the Adapters structure described in Section "Adapters design". Let us take the typical case where adapters are employed to correct convolutional layers. Let us say $w_{f_n} \in \mathbb{R}^{K_n \times K_n \times I_n \times O_n}$, where $K_n \times K_n \times I_n$ is the single filter size and O_n is the number of filters in the n-th layer. Please note that an extra O_n term may come from biases that are typically absent if batch normalization, as in our case, follows the convolutional layer.

We know that the space complexity of this layer is $\Theta(K_n \times K_n \times I_n \times O_n)$. In our design, the adapters are constituted of a batch normalization layer having $\Theta(4 \cdot O_n)$(as it includes a set of four parameters per channel) and the 1×1 convolution layer has $\Theta(O_n)$. Overall, we know that each adapter has space complexity $\Theta(5 \cdot O_n)$ of parameters to be updated, in place of $\Theta[O_n(K_n \times K_n \times I_n + 5)]$(that include both f_n and BN_n): evidently, for $K_n > 1$ and $I_n > 1$(which is the typical case) we have a considerable parameter complexity saving, even at train time. This comes, however, at the cost of extra computation at inference time. In principle, this is avoidable by layer folding, possible given that in the proposed design we have included no non-linearity between f_n and A_n: this is left for further studies.

Adapter fusion

A way to decrease memory consumption and, simultaneously, the computational cost is to compress the architecture by fusing the adapter batch normalization layers and the sum operation with the adapter 1×1 convolutional layer (Step 1). After the adapter is fused, the newly obtained 1×1 convolutional layer is fused with the batch norm layer of the original architecture (Step 2). Finally, the 1×1 convolutional layer is fused into the backbone's original 3×3 convolutional layer (Step 3). Figure 3 visualizes how to fuse these layers step-by-step.

Step 1. For the sake of clarity, we simplify the adapter operation defined in (6) to focus on the important aspects of the compression. Let f_n be the output of the n-th layer before the adapter, f_{A_n} be a 1×1 convolutional operation and BN_{A_n} be a batch norm operation. The output of the layer following the adapter, indicated as $\tilde{f}_n$ is equal to

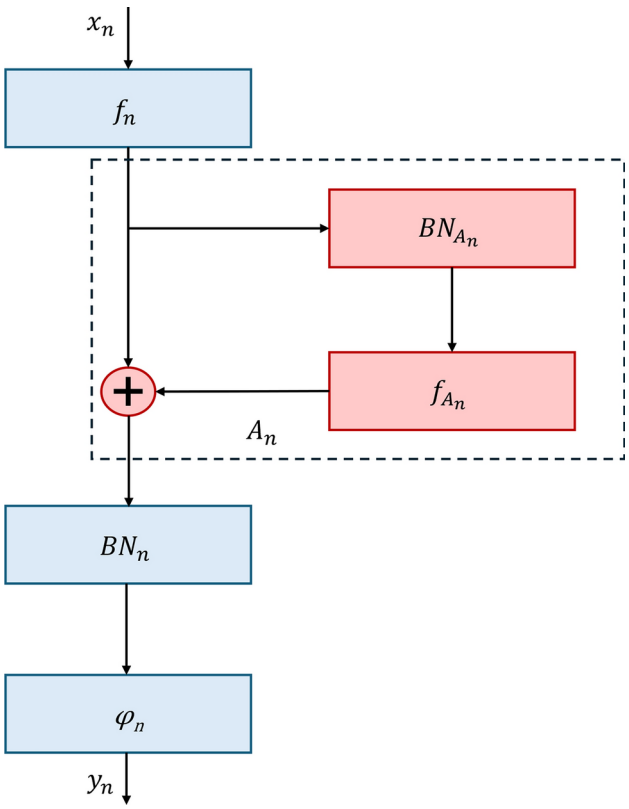
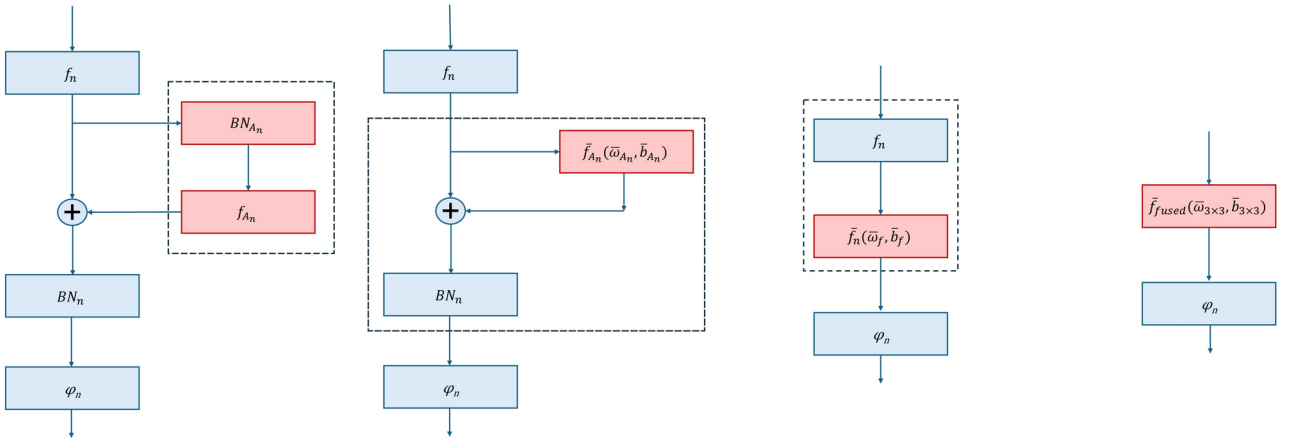


Fig. 2. A schematic showing how adapters (A_n, in red) are plugged into the rest of the architecture (blue).



(a) Step 1. **(b)** Step 2. **(c)** Step 3. **(d)** Final result.

Fig. 3. Adapter fusion scheme, visualized step-by-step.

$$\tilde{f}_n = f_n \; + \; f_{A_n}\left[\mathrm{BN}_{A_n}(f_n)\right]. \tag{7}$$

We will start by detailing the matrix operations behind the batch norm and convolutional layers. Let $w_{\mathrm{BN}_{A_n}}$ and $b_{\mathrm{BN}_{A_n}}$ be weights and biases of shape $\mathbb{R}^C$, where C is the number of features of previous layers. The batch norm layer computes the running mean $\mathbb{E}[x]$, and running standard deviation $\sigma[x]$, where x is the output of the previous layer. Then, it performs a scaling and an element-wise multiplication, which we indicate with the symbol $\odot$. The forward operation of a batch norm layer can be defined as

$$\mathrm{BN}(x) = \frac{x - \mathbb{E}[x]}{\sigma[x]} \odot w_{\mathrm{BN}_{A_n}} + b_{\mathrm{BN}_{A_n}}. \tag{8}$$

For the convolutional layers, we use $*$ to indicate the convolutional operation, $w_{f_{A_n}}$ convolutional weights and $b_{f_{A_n}}$ the bias. With x being the output of previous layers and f indicating the convolutional operation, we can define the convolutional forward operation as

$$f(x) = w_{f_{A_n}} * x + b_{f_{A_n}}. \tag{9}$$

We can now put all the operations together: replacing (8) and (9) in (7), we have

$$\tilde{f}_n = f_n + w_{f_{A_n}} * \left[\frac{f_n - \mathbb{E}[f_n]}{\sigma[f_n]} \odot w_{\mathrm{BN}_{A_n}} + b_{\mathrm{BN}_{A_n}}\right] + b_{f_{A_n}}. \tag{10}$$

Now, let $\nVdash$ be an identity matrix of shape equal to w_{A_n} and $\times$ the matrix multiplication symbol. We can now define two equivalent weights and biases to compress our operation, which will be indicated respectively as $\tilde{w}_{A_n}$ and $\tilde{b}_{A_n}$

$$\begin{aligned} \tilde{w}_{A_n} &= \nVdash + w_{f_{A_n}} \odot \frac{w_{\mathrm{BN}_{A_n}}}{\sigma[f_n]} \\ \tilde{b}_{A_n} &= w_{\mathrm{BN}_{A_n}} \times \left(b_{\mathrm{BN}_{A_n}} - \frac{w_{\mathrm{BN}_{A_n}} \odot \mathbb{E}[f_n]}{\sigma[f_n]}\right) + b_{f_{A_n}}. \end{aligned} \tag{11}$$

Step 2. Now, after the adapter compression, we find ourselves with a 3×3 convolutional layer, a 1×1 convolutional layer, and a batch norm layer. To avoid confusion, let us indicate x as the output of the 3×3 convolutional operation and f as the 1×1 convolutional operation. We use w_f and b_f for the convolutional weights and biases, w_{BN} and b_{BN} for the batch norm weights and biases, $\mathbb{E}[f(x)]$ and $\sigma[f(x)]$ for running averages and standard deviations. The sequence of the two operations yields

$$\mathrm{BN}(f(x)) = \frac{w_f \odot w_{\mathrm{BN}}}{\sigma[f(x)]} \odot x + w_{\mathrm{BN}} \times \frac{b_f - \mathbb{E}[f(x)]}{\sigma[f(x)]} + b_{\mathrm{BN}}. \tag{12}$$

Therefore, we can define new fused weights and biases as

$$\tilde{w}_f = \frac{w_f \odot w_{\mathrm{BN}}}{\sigma[f(x)]}, \quad \tilde{b}_f = w_{\mathrm{BN}} \times \frac{b_f - \mathbb{E}[f(x)]}{\sigma[f(x)]} + b_{\mathrm{BN}}. \tag{13}$$

Step 3. Finally, we have a 3×3 convolutional layer and a 1×1 convolutional layer. Weights and biases of the former are represented by $w_{3\times3}$ and $b_{3\times3}$, whereas the latter ones are $w_{1\times1}$ and $b_{1\times1}$. We indicate with C_{out} the output channel and C_{in} the input channel. Therefore, we have $w_{3\times3} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times 3 \times 3}$ and $b_{3\times3} \in \mathbb{R}^{C_{\text{out}}}$, while $w_{1\times1} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ and $b_{1\times1} \in \mathbb{R}^{C_{\text{out}}}$. Each output channel $o \in [0, C_{out} - 1]$ of the fused weight $\tilde{w}_{3\times3}$ and bias $b_{3\times3}$ can be computed as

$$\tilde{w}_{3\times3}[o,:,:,:] = \sum_{m=0}^{C_{out}} w_{1x1}[o,m] \cdot w_{3x3}[m,:,:,:], \quad \tilde{b}_{3\times3}[o] = b_{1x1}[o] + \sum_{m=0}^{C_{out}} w_{1x1}[o,m] \cdot b_{3x3}[m]. \tag{14}$$

This "fusion" procedure helps the adapted-equipped architecture, eliminating the operational cost of the adapter introduction, namely the new layers insertion. It happens in two ways: first of all, the adapter fusion reduces the total number of layers, lowering the number of operations (i.e. FLOPs) of the model and eliminating the FLOPs related to the adapter insertion. Furthermore, the adapter's size is absorbed within the architecture, zeroing its memory size cost. Finally, performing the fusion returns the model structure to the starting one, making new modifications feasible and simpler.

Adapters ranking

In this section, we propose a method for ranking adapters and assessing whether they are worth keeping in the architecture or can be discarded for memory-saving reasons.

Many works focusing on removing parameters from over-parametrized models[31,43–46] propose simple magnitude-based schemes (which can naively be seen as norm ranking schemes to determine which are the least relevant parameters). When training a model with regularization like ℓ_1 or even ℓ_2 it is empirically validated, in a multitude of setups, that parameters in excess are likely to have low magnitude, and simple schemes like norm ranking constitute a robust baseline[47,48].

However, it is also known that global pruning strategies are impossible to deploy in the wild, due to different norm scaling in DNN's layers[49,50]. Indeed, the parameter's average magnitude varies layer-by-layer, depending on a multitude of factors that include initialization[51,52] and the average backpropagation signal received[53] to name a few. One fair strategy would consist of ranking the adapters by their number of parameters (which would be the ℓ_0 norm), motivated by the more significant potential expressivity of these.

We argue that both the average parameter's norm and the adapter's cardinality (intended as the number of its parameters) should be accounted for, and we propose a metric estimating the average magnitude-per-parameter

$$Z = \frac{\|w_{f_n}\|^2}{|w_{f_n}|}, \tag{15}$$

where Z is the relative score for the current layer, $\|\cdot\|$ is the ℓ_2 norm, and $|\cdot|$ is the parameter's cardinality. We expect the larger the value in (15), the higher the average norm per parameter, meaning that the current adapter is relevantly contributing to correcting the layer's output and should be considered "important". On the contrary, a low value of Z indicates little perturbation of the layer's output and can be considered insignificant.

We exclude higher-order evaluation schemes for two reasons. The first one is that such an evaluation will be performed post-hoc, after training, meaning that we can expect first-order derivatives to be approximately zero, making first-order evaluation methods like[54,55] not usable. In such context, higher-order methods should be used, which, however, are computationally expensive: to maintain their complexity at bay, approximations are introduced, making them unreliable in high-dimensionality [56–59]. The second reason is that some of these methods introduce extra regularization constraints to make the model's pruning compliant with the pruning scheme[55,59]. At the same time, we want to remain as agnostic as possible to the training scheme, requiring simple ℓ_1 or ℓ_2 regularization schemes.

By quantifying the significance of adapter modules through Z we can establish a ranking among them. This enables a trade-off strategy in which only the most relevant adapters are kept, while the least significant ones are discarded. Such a selection process requires a little cost in performance, as the removed adapters contribute little to the model's output, while simultaneously reducing the overall memory footprint of the adapter set.

Experiments

In this section, we provide details about the baseline architecture, the adaptation procedures explored in new domains, and the experiments we conducted with the adapter modules and memory-saving strategies, reporting the results we obtained.

Setup

Data. The studied datasets are three, provided by NASA's open-source database. The first one, called *Synthetic Moon* dataset (shortly *SMo*), contains 9.766 realistic artificial renders of the lunar landscape, together with 36 real photos of the Moon's environment, *Real Moon* or *RMo*, (see[60] for more details about this dataset). The other two regarding Mars are named *AI4Mars*[17] and *MarsData-V2*[61] and consist of real photographs of the Mars landscape, 18.1k and 835 images respectively. We classified these data into three common classes: rocks, to be avoided by the rovers, safe terrain, to move upon, and finally a label identifying sky. Further details on the dataset and the data preprocessing can be found in the Supplementary Material.

Architecture.

The chosen base architecture is U-Net[62] due to its well-known high-performing feature extractor modules, the versatility of segmentation tasks, and the reduced training time. We performed a grid search to determine the

best data augmentation techniques, loss function, and optimizer. We found the best solution for performing data augmentation with Random Crop (p = 0.50%) on training images. All architectures have been trained using a Balanced Categorical Cross Entropy loss (with weighted classes). The chosen optimizer was *Adam* with a starting learning rate of 0.001. The learning rate was then reduced by a factor of 10 every time the validation loss didn't improve for 10 epochs until it reached 10^{-6}.

We also tested different encoders, from the typical ResNet family[63], the traditional Vgg (version with batch norm layers[64]), and also a version of the EfficientNet[65]. Furthermore, we explored two light architectures devoted to smaller devices, like the MobileNetv2[66] and the lightest version of MobileOne[67], discarding more complex ones like the DenseNets[68], due to their higher computational cost, making them unsuitable for space applications.

For the same reasons, although their elevated performance, we didn't study the Transformer[69] architecture, because of its computational costs and memory size.

All the implementation details are provided in the Supplementary Material.

Main results

In our experiments, we include the following strategies.

- *Baseline*: use the pre-trained neural network on *SMo* to predict class labels on adaptation domains.
- *Scratch*: training from scratch a new neural network on adaptation domains.
- *Full finetuning*: fine-tuning both the encoder and decoder of the baseline neural network.
- *Encoder finetuning*: fine-tuning just the encoder.
- *Decoder finetuning*: fine-tuning just the decoder.
- *Batchnorm finetuning*: training batchnorm layers and keeping everything else frozen.
- *Adapters (All)*: training adapter modules and freezing the rest of the pre-trained neural network.
- *Adapters (Ranked and fused)*: selecting the most important adapters with the ranking method and fuse them in the original architecture.

The first way consisted of the study of different grades of re-training: a complete one from scratch, maintaining just the same architecture (called *Baseline*), a complete finetuning of the synthetic lunar model, adjusting the weights on the new domain (the *Full finetuning* model) and then the finetuning of just some layers of the initial model (*Encoder*, *Decoder* and *Batchnorm*). On the other hand, by introducing adapters, we studied the case of training just the adapters on the new domain while freezing the network on the initial synthetic lunar weights. Furthermore, the measurements of the application of the new strategies are reported too; their implementation will be detailed in the following sections. Together with the performance differences between the algorithms, the p-values are also provided. They are evaluated using a paired t-test or a Wilcoxon test to determine whether the values are normally distributed. The difference is statistically significant if $p < 0.05$. The results reported in Table 2 show not only the performances of these different adaptation strategies together with the FLOPs counting but also the trained parameters (and their ratio on the total ones) as a comparison. It can be observed that the adapter method has an extreme gain in the number of trained parameters with a relatively small increase in the number of operations (FLOPs) and slightly worse performance. At the cost of $\approx 10\%$ of trained parameters, a similar level of performance of a completely re-tuned model can be achieved.

This behavior can also be visualized in Pareto curves (Fig. 4) showing the mean balanced accuracy versus the number of trained parameters. As can be seen, the top left corner of the plots corresponds to adapters, meaning that high performances can be reached with a relatively small training price. Notice that the architecture of reported adapters in Table 2 is a module composed of a BN layer connected to a convolutional 1×1 one. Different structures were studied and their performances are presented in Section "Adapters architecture".

Considering each domain specifically, the best choice for real lunar images is to keep the pre-trained weights from the synthetic lunar model and finetune them to adjust to the new domain. The adapter application helps with the memory issue at the cost of a slightly worse performance($\approx 3\%$, statistically significant $p = 0.02$). The same considerations can be made for the Martian domains, especially about fine-tuning. Due to the obvious differences between the lunar and Martian environments, it's not mandatory to keep the already performed training. On the other hand, the adapter introduction brings the same advantages as before, reaching even nearer levels of performance to the more completed retrained models. In particular, the AI4Mars's adapters implemented in the Vgg19-BN lose just $\approx 1.2\%$($p = 1.1e-5$) more than the best method, i.e., the encoder finetuning one. In contrast, the MarsDataset ones perform better with ResNet-18 encoder, worsening the full finetuning performance by only $\approx 1.9\%$($p = 2.3e-5$). A prediction example with a comparison between adaptation methods on MarsDataset and RealMoon domains can be seen in Fig. 5.

A preliminary noise robustness study, where the baseline model and the adapter equipped one performance are compared under different noise conditions, is presented in the Supplementary Material.

Adapters architecture

We compared here different adapter configurations, visualized in the Supplementary Material. In Table 3 the results of their performance on the RMo dataset (validation set) are presented. Interestingly, introducing a non-linearity such as a ReLU function didn't bring any improvement, validating that the best design results in the sequence of a BN layer followed by a convolutional 1×1 one. We infer this is due to the tight relationship between domains, where simple linear corrections at the layer's scale are sufficient to improve performance dramatically. Employing higher complexity makes the optimization task harder, meaning that finding the optimal training policy is much harder - this justifies the performance drop in those cases.

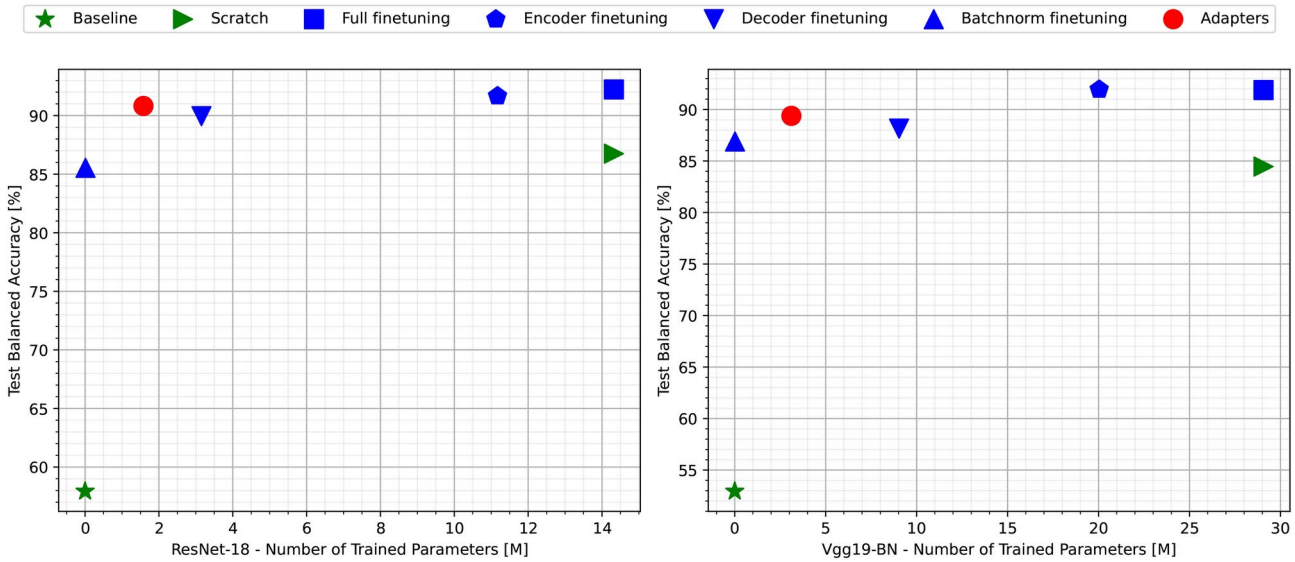


Fig. 4. Pareto curves for VGG19-BN and ResNet-18.

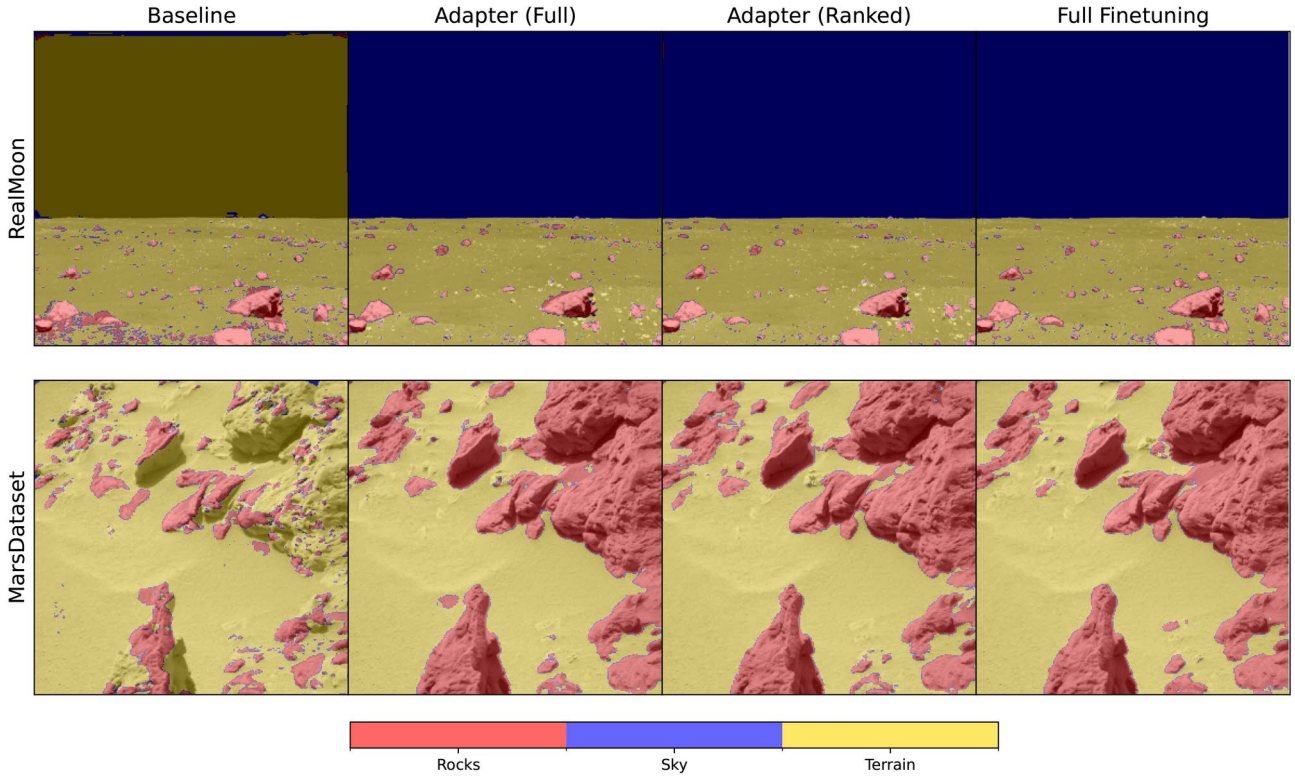


Fig. 5. Comparison of U-Net predictions on real Moon and real Mars images using different methods.

Adapters ranking

Subsequently to the adapter introduction, as described in Section "Adapters ranking", we perform an adapter evaluation, ranking them from the most significant to the least one through different scores. The Pareto curves (Fig. 6) show the balanced accuracy behaviour as a function of the cumulative size of the selected adapter up to the total adapter set. From them, it's clearly visible that the third score, the square norm normalized by the layer size (see Section "Adapters ranking"), represented in blue, provides the best adapters ranking, reaching first the best performance at the smaller cumulative size. To select the best trade-off in the adapter removal, we accept a worsening of 0.5% from the top performance (validation-wise), cutting away the other less significant adapters and saving memory. For example, for MarsDataset we noticed that only 18 adapters out of the 27 total adapters

Adapter design				
BN	ReLU	Conv1x1	BN	Balanced accuracy (%)
✓		✓		91.42
✓	✓	✓		89.66
		✓	✓	75.84

Table 3. Ablation on different adapter designs on *RMo*.

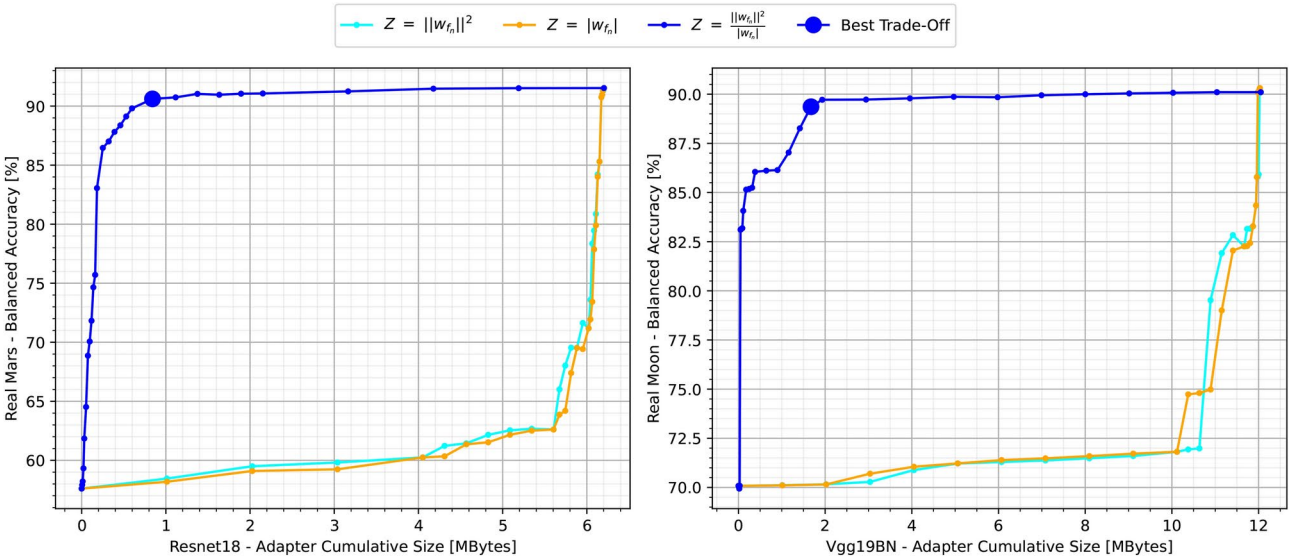


Fig. 6. Performance as a function of the cumulative size of inserted adapters: left and right images show respectively the results on real mars (MarsDataset) and Real Moon domains (test sets).

are truly necessary for the Resnet-18 U-Net. By using only 18 adapters we have a slight decrease in performance ($\approx -1.0\%$ max), but a huge decrease in memory consumption: instead of all of the adapter set, in terms of parameters, just the 8.2%(Resnet-18) and 15.4%(Vgg19-BN) of them can be used (see Table 2).

The position of the removed modules is also quite interesting: these turn out to be the heaviest ones (size-wise) and, due to the U-Net architecture, they are placed in the deepest part of the architecture. This means that the deeper the block, the less important it is, implying that the most important adapters for feature extraction and classification are in the first encoder layers and the last decoder layers. A scheme of these can be seen in Fig. 7 (see Supplementary Material for Vgg). Notice that to perform this strategy, it is still necessary to train all adapters and then proceed with measuring their ranks and, subsequently, the selection. For this reason, in Table 2 the trained parameters are presented in double mode: all of the adapter parameters are effectively trained and just then the ranking pruning is applied, cutting massively their numbers. In the sense of a practical application, these results mean that we can reduce the update memory size for a different domain using the adapter modules, and we can also further save memory by sending just a fraction of the total number of adapters, the most significant ones, with minimal performance loss.

Adapters fusion

In the end, employing the adapters fusion strategy thoroughly described in Section "Adapter fusion", we produce a "fused" version of the previously adapted models, absorbing the ranked adapters inside the baseline architecture. Applying this process decreases also the computational number of operations (i.e. FLOPs) while keeping the same level of performance (average $p = 0.59$, differences extremely not statistically significant).

The "fusion" procedure doesn't affect the performance. Still, the main gain comes in combination with the ranking one: while the ranking strategy prunes the number of required adapters, saving a huge amount of memory, the fusion one simplifies the architecture, decreasing the overall FLOPs and zeroing the cost of the adapters introduction. Furthermore, the "fused" model now has the same structure as the initial one, making other updates, modifications, or module insertions (like other future adapters) more feasible.

The values can be seen in Table 2: while just the ranking strategy worsens the performance, the model size and the FLOPs number return to the baseline models' level, decreasing both of them by $\approx 10\%$ than utilizing all of the adapter set. Notice also that the storage memory of the fused architecture reaches the same level as the baseline one (see also Table 1), proving that in the end, we will have nearly similar architectures, but now with the changed adapted parameters.

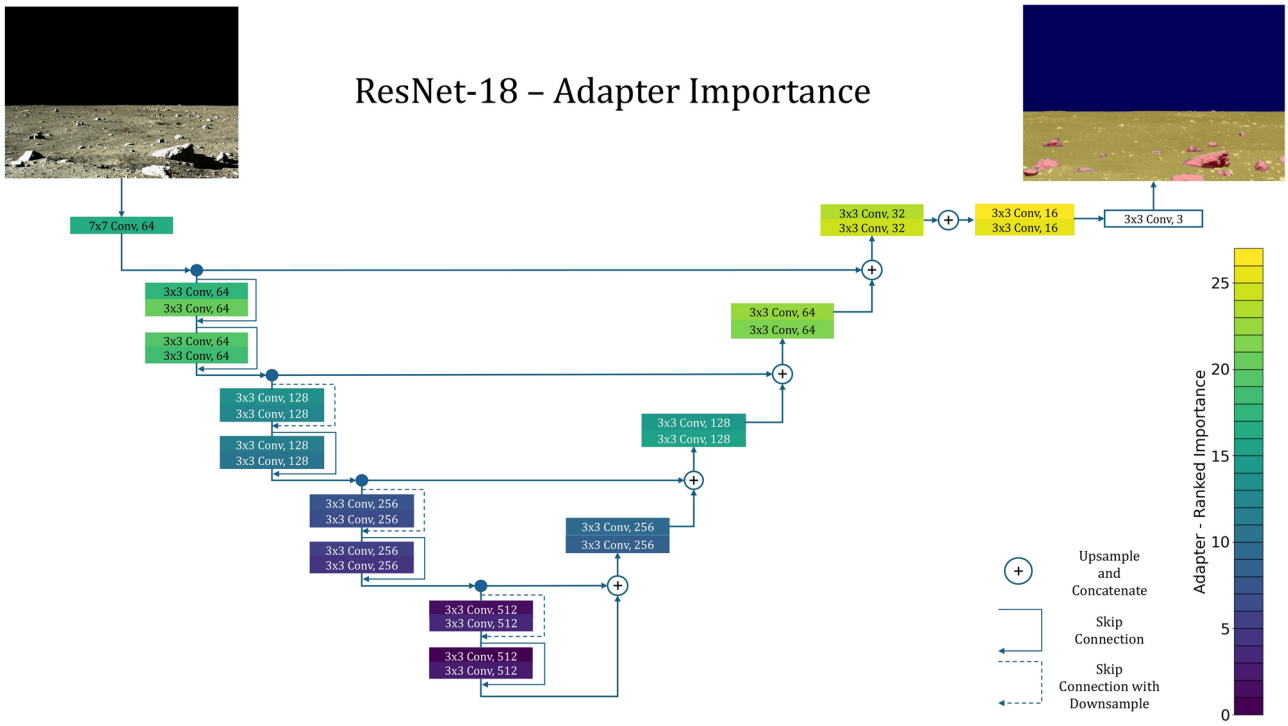


Fig. 7. Visualization of Resnet-18 adapter ranking: modules near the bottleneck of the U-Net and with larger size are less important than the ones in first encoder layers and last decoder layers.

Encoder	Total params [M]	FLOPs [G]	Balanced accuracy [%]		IoU [%]		Storage memory [MB]	Compressed (.tar.gz) [MB]
			Validation	Test	Validation	Test		
ResNet-18	14.32	21.42	96.49	94.98	83.76	82.32	54.75	50.75
ResNet-34	24.43	31.12	96.46	94.89	82.93	82.05	93.37	86.44
ResNet-50	32.52	42.58	96.35	94.96	83.37	81.99	124.39	115.20
EfficientNet-b3	13.16	15.75	96.16	94.77	85.28	83.10	50.79	47.08
MobileOne-s0	8.59	18.95	90.99	92.40	76.95	79.35	33.62	30.80
MobileNet-v2	6.63	13.69	94.89	93.45	77.43	77.63	25.57	23.61
Vgg11-BN	18.26	57.94	96.09	94.73	82.57	81.29	69.74	64.60
Vgg13-BN	18.44	77.39	96.16	94.76	82.68	81.24	70.45	65.35
Vgg16-BN	23.76	99.17	96.46	95.00	83.28	82.11	90.73	84.21
Vgg19-BN	29.07	120.94	96.58	95.16	83.48	82.18	111.01	103.05

Table 1. U-Net encoders study on the Synthetic Moon dataset.

Inference on embedded devices

As previously shown, the main advantage of the adapter's implementation is the flexibility to adapt to a new domain, keeping the same level of performance. Memory is one major factor to consider for embedded devices, due to their limited resources. The hardware cannot handle a complete re-training by itself, so a lighter approach, such as a small update from Earth, can keep the performance on level. This gain might be negligible if the computational training power can manage a completely new training, but often the on-field devices are not able to do so, performing just the inference phase of the trained model.

Due to these limitations, it's fundamental to measure the impact of the adapter on the model, especially regarding the inference time and the memory usage, while loaded on an embedded device. To simulate an on-the-field situation, we decided to test two different hardware: a Raspberry Pi 4 board, equipped only with a CPU processor, and a Jetson Orin Nano one, with a Nvidia GPU, measuring the inference time, the memory usage, and FLOPs numbers while working on board. This choice allowed us to compare the different performances between the two hardware (CPU - GPU) of our models. It is worth noting that, although GPUs have not been implemented yet on on-site rovers, the increase in the onboard computing performance demands might accelerate the usage of such devices in space[70], which is the reason why we also included a GPU device in our inference experiments. Further details on the two devices can be found in the Supplementary Material.

Method	Encoder	Trained params (M)	Trained params (%)	Forward FLOPs (G)	Balanced Accuracy (%)			
					RMo	AI4Mars	Marsdataset	Mean
Baseline	ResNet-18	14.32	100.00	21.42	82.14	34.05	57.61	57.93
Scratch	ResNet-18	14.32	100.00	21.42	85.41	87.89	86.96	86.75
Full finetuning	ResNet-18	14.32	100.00	21.42	92.70	90.53	93.46	**92.23**
Encoder finetuning	ResNet-18	11.17	77.99	21.42	90.84	91.80	92.36	91.67
Decoder finetuning	ResNet-18	3.15	22.01	21.42	90.03	86.35	93.46	89.95
Batchnorm finetuning	ResNet-18	0.01	0.007	21.42	85.44	83.00	88.17	85.54
Adapters (All)	ResNet-18	1.58	9.94	23.57	89.68	89.99	91.53	90.82
Adapters (Ranked and fused)	ResNet-18	0.13	0.82	21.27	89.48	80.30	90.61	86.80
Baseline	Vgg19-BN	29.07	100.00	120.94	70.08	35.21	53.54	52.94
Scratch	Vgg19-BN	29.07	100.00	120.94	82.76	86.71	83.92	84.46
Full finetuning	Vgg19-BN	29.07	100.00	120.94	93.65	90.34	91.64	91.88
Encoder finetuning	Vgg19-BN	20.03	68.92	120.94	90.48	**91.93**	93.41	91.94
Decoder finetuning	Vgg19-BN	9.03	31.08	120.94	89.14	85.97	89.30	88.14
Batchnorm finetuning	Vgg19-BN	0.02	0.007	120.94	88.43	81.26	90.98	86.89
Adapters (All)	Vgg19-BN	3.11	9.68	136.18	90.10	90.69	87.96	89.38
Adapters (Ranked and fused)	Vgg19-BN	0.48	1.49	120.47	89.36	86.18	87.81	87.78

Table 2. Performance on adapted domains: Moon and Mars. Ranked and fused lines are given in bold.

Encoder	Adapter type	Balanced accuracy [%]	FLOPs [G]		Storage memory [MB]		Compressed (.tar.gz) [MB]		Energy spent [kJ]	Inference [s]	
			Total	Update	Total	Update	Total	Update		RPi4	JNano
ResNet-18	None	93.46	21.42	+21.42	54.79	+ 54.79	50.78	+ 50.78	39.36	6.96	0.60
	All	91.53	23.57	+2.15	60.91	+ 6.12	56.42	+ 5.64	27.28	8.29	0.77
	Ranked	90.61	22.96	+1.54	55.61	+ 0.82	51.54	+ 0.76	–	7.51	0.69
	Fused	**90.61**	**21.27**	**-0.15**	**54.67**	**- 0.08**	**50.80**	**+ 0.02**	–	**7.05**	**0.63**
Vgg-19-BN	None	93.65	120.94	+120.94	111.01	+ 111.01	55.00	+ 55.00	2.37	25.78	0.95
	All	90.10	136.18	+15.24	123.00	+ 11.99	66.04	+ 11.04	2.36	33.26	1.45
	Ranked	89.36	131.79	+10.85	112.72	+ 1.71	56.57	+ 1.57	–	28.12	1.11
	Fused	**89.36**	**120.47**	**-0.47**	**110.88**	**- 0.13**	**103.03**	**+ 48.03**	–	**25.86**	**0.97**

Table 4. Final memory and computational cost comparison. "None" means the finetuning of the baseline model without the adapters. "Ranked", "Fused" and "Energy costs" values are dataset-specific (ResNet-18 - MarsDataset, Vgg19-BN - RMo).

A comprehensive collection of the performances of the previously presented models is also reported in the Material, while Table 4 reports a summary of performance and computational effort (in terms of FLOPs and inference time). The adapter usage worsens the performance and increases the total model size and complexity, but the update cost is drastically reduced. If the two new proposed strategies are performed, even these costs are zeroed out: memory footprint, storage size, and computational cost (i.e., FLOPs and inference time) are returned to baseline level, at the cost of even less updated memory dimension.

Discussion

In Table 4 we reported the final comparison between the presented steps, following also a hypothetical order of procedure. A hypothetical application is a situation of an on-field device equipped with a DL model for rock segmentation (for example for autonomous driving). Given such devices' computational and hardware constraints, both the baseline training and subsequent model updates must be performed at ground stations before being transmitted to the deployed system.

For instance, in a lunar environment, the baseline is trained over a synthetic dataset providing a set of generic features (e.g. *SMo*). Then, we make the case that upgrades are needed because of the change in the environmental conditions: the model should be adapted to a new domain that is similar but at the same time different from the baseline starting one. Note that this adjustment is possible only if a large enough dataset of the new environment is already provided.

First of all, the simplest solution consists of a completely new finetuning and adjustment of the parameters to the new domain. The *None* lines show the cost of a complete finetuning in terms of FLOPs and Storage Memory, considering re-training 100% of the model without the use of the adapter modules. Doing so requires re-sending the full-finetuned model entirely to the devices (see Storage Memory update size equals to the total one), making difficult the update due to the bandwidth constraints, for example.

Instead, we presented the advantages of the adapter's usage in this work. As a first approach, an entire set of the trained-on-new-domain adapters can be used to update the model (namely, the *All* option). While this approach marginally increases computational costs (e.g., $+2.15$ GFLOPs), it significantly reduces the memory required for updates (e.g., $+6.12$ MB). Additionally, training only the adapters requires less energy.

These benefits can be further improved: thanks to the "adapter ranking" strategy just a tiny subset of the adapter set can be sent (see *Ranked*), reducing even more the memory cost (e.g. $+0.82$ MB) and limiting a little bit the computational operations (e.g. $+1.54$ GFLOPs). However, this reduction comes at the expense of a slight worsening of performance (see Section "Adapters ranking"), it could provide a viable trade-off, matching the space exploration constraints. Finally, these selected adapters can be absorbed within the architecture by the "adapter fusion" strategy (*Fused*): this results in a model with computational and storage costs nearly identical to the original baseline, while still being adapted to the new domain with minimal performance loss. Importantly, the fusion operation can be performed directly on the device, meaning the effective update cost remains the same as the *Ranked* configuration. However, the computational overhead introduced by adapters is eliminated, simplifying the model architecture and facilitating future updates. The memory saving on the update size using these strategies fits well with the usual bandwidth limitations in space exploration, together with a low computational cost, due to the space-certified hardware characteristics.

Conclusion

In this work, we studied the rock segmentation problem on real photos of the Moon and Mars exploiting the adapter modules with two different memory-saving strategies. We show that adapters have advantages over performing a complete re-training of the base architecture for each dataset regarding the space exploration constraints in bandwidth communication and flexibility. We obtain segmentation performances comparable to complete retrained models with adapters ($\approx 10\%$ of the parameters). We also showed two techniques to exploit these advantages further. *Adapters ranking* is the first, showing that just a smaller portion of adapters is genuinely significant, pruning some of them and saving more memory ($\approx 86\%$), at a slight cost in performance ($\approx 0.5\%$). Then, *adapter fusion* fits them into the architecture and lowers the computational cost ($\approx 10\%$). The memory saving is less significant ($< 10\%$), but this strategy keeps the architecture simpler and faster than before, making other modifications more feasible. We are confident these findings will enable research in resource-constrained environments, where computation is critical and where bandwidth for updates is very limited.

Data availability

The data used in this article comes from publicly available resources. The datasets used during the current study are available for the Synthetic Moon dataset, AI4Mars, and Marsdataset. LO and ESM re-annotated Marsdataset, where the new ground-truths can be found at https://github.com/lolivi/MoonMarsAdapters/tree/main/marsdataset-sky-annotations.

Received: 31 December 2024; Accepted: 17 April 2025
Published online: 23 May 2025

References

1. Ge, D., Cui, P. & Zhu, S. Recent development of autonomous GNC technologies for small celestial body descent and landing. *Progress Aerospace Sci.***110** (2019).
2. Gor, V., Mjolsness, E., Manduchi, R., Castano, R. & Anderson, R. Autonomous rock detection for mars terrain. In *AIAA Space 2001 Conference and Exposition* (2001).
3. Furlán, F., Rubio, E., Sossa, H. & Ponce, V. Rock detection in a mars-like environment using a cnn. In *Pattern Recognition: 11th Mexican Conference, MCPR 2019, Querétaro, Mexico, June 26–29, 2019, Proceedings 11* (Springer, 2019).
4. Rebuffi, S.-A., Bilen, H. & Vedaldi, A. Learning multiple visual domains with residual adapters. *Adv. Neural Inf. Process. Syst.***30** (2017).
5. Thompson, D. R. & Castano, R. Performance comparison of rock detection algorithms for autonomous planetary geology. In *2007 IEEE Aerospace Conference* (IEEE, 2007).
6. Castano, R. et al. Current results from a rover science data analysis system. In *2005 IEEE Aerospace Conference* (IEEE, 2005).
7. Ono, M., Fuchs, T. J., Steffy, A., Maimone, M. & Yen, J. Risk-aware planetary rover operation: Autonomous terrain classification and path planning. In *2015 IEEE Aerospace Conference* (IEEE, 2015).
8. Burl, M. C., Thompson, D. R., deGranville, C. & Bornstein, B. J. Rockster: Onboard rock segmentation through edge regrouping. *J. Aerospace Inf. Syst.***13** (2016).
9. Xiao, X., Cui, H., Yao, M. & Tian, Y. Autonomous rock detection on mars through region contrast. *Adv. Space Res.***60** (2017).
10. Gong, X. & Liu, J. Rock detection via superpixel graph cuts. In *2012 19th IEEE International Conference on Image Processing* (IEEE, 2012).
11. Golombek, M., Huertas, A., Kipp, D. & Calef, F. Detection and characterization of rocks and rock size-frequency distributions at the final four mars science laboratory landing sites. *Int. J. Mars Sci. Explor.***7** (2012).
12. Dunlop, H., Thompson, D. R. & Wettergreen, D. Multi-scale features for detection and segmentation of rocks in mars images. In *2007 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2007).
13. Kuang, B., Wisniewski, M., Rana, Z. A. & Zhao, Y. Rock segmentation in the navigation vision of the planetary rovers. *Mathematics***9** (2021).
14. Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N. & Liang, J. Unet++: Redesigning skip connections to exploit multiscale features in image segmentation. *IEEE Trans. Med. Imaging***39** (2019).
15. Wagstaff, K. L. et al. Deep mars: CNN classification of mars imagery for the PDS imaging atlas. In *IAAI* (2018).
16. Rothrock, B. et al. Spoc: Deep learning-based terrain classification for mars rover missions. In *AIAA SPACE* (2016).
17. Swan, R. M. et al. AI4MARS: A dataset for terrain-aware autonomous driving on mars. In *CVPRW* (2021).
18. Chen, L., Zhu, Y., Papandreou, G., Schroff, F. & Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV* (2018).
19. Liu, H., Yao, M., Xiao, X. & Xiong, Y. Rockformer: A u-shaped transformer network for martian rock segmentation. *IEEE Trans. Geosci. Remote Sensing***61** (2023).

20. Xiong, Y. et al. Marsformer: Martian rock semantic segmentation with transformer. *TGRS* (2023).
21. Zhang, J., Xia, Y. & Shen, G. A novel deep neural network architecture for mars visual navigation. (2018).
22. Marek, D. & Nalepa, J. End-to-end deep learning pipeline for on-board extraterrestrial rock segmentation. *Eng. Appl. Artif. Intell.* **127**, 107311. https://doi.org/10.1016/j.engappai.2023.107311 (2024).
23. Pan, S. J. & Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.***22** (2009).
24. Li, J. et al. Autonomous martian rock image classification based on transfer deep learning methods. *Earth Sci. Inf.***13** (2020).
25. Sargano, A. B., Wang, X., Angelov, P. & Habib, Z. Human action recognition using transfer learning with deep representations. In *2017 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2017).
26. Zhao, H., Liu, Q. & Yang, Y. Transfer learning with ensemble of multiple feature representations. In *2018 IEEE 16th International Conference On Software Engineering Research, Management and Applications (SERA)* (IEEE, 2018).
27. Quétu, V. & Tartaglione, E. Dsd2: Can we dodge sparse double descent and compress the neural network worry-free? In *Proceedings of the AAAI Conference on Artificial Intelligence* (2024).
28. Wang, D., Li, Y., Lin, Y. & Zhuang, Y. Relational knowledge transfer for zero-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30 (2016).
29. Wang, S. & Li, Z. A new transfer learning boosting approach based on distribution measure with an application on facial expression recognition. In *2014 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2014).
30. Wang, T., Huan, J. & Zhu, M. Instance-based deep transfer learning. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE, 2019).
31. Marouf, I. E., Tartaglione, E. & Lathuilière, S. Mini but mighty: Finetuning vits with mini adapters. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2024).
32. Furano, G. et al. Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities. *IEEE Aerosp. Electron. Syst. Mag.* **35**, 44–56 (2020).
33. Ziaja, M. et al. Benchmarking deep learning for on-board space applications. *Remote Sensing* **13**, 3981 (2021).
34. Nguyen, L.-T., Quélennec, A., Tartaglione, E., Tardieu, S. & Nguyen, V.-T. Activation map compression through tensor decomposition for deep learning.
35. Han, S., Mao, H. & Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. *Comput. Vis. Pattern Recogn.* (2015).
36. Najafabadi, M. M. et al. Deep learning applications and challenges in big data analytics. *J. Big Data***2** (2015).
37. Dhar, S. et al. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Trans. Int. Things* **2**, 1–49 (2021).
38. Rajapakse, V., Karunanayake, I. & Ahmed, N. Intelligence at the extreme edge: A survey on reformable tinyml. *ACM Comput. Surv.* **55**, 1–30 (2023).
39. Chien, S. et al. Agile science for primitive bodies and deep space exploration, https://doi.org/10.2514/6.2014-1888.
40. Nalepa, J. et al. Towards on-board hyperspectral satellite image segmentation: Understanding robustness of deep learning through simulating acquisition conditions. *Remote Sensing* https://doi.org/10.3390/rs13081532 (2021).
41. Bamford, T., Esmaeili, K. & Schoellig, A. P. A deep learning approach for rock fragmentation analysis. *Int. J. Rock Mech. Min. Sci.* **145**, 104839 (2021).
42. Rebuffi, S., Bilen, H. & Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. *CoRR*. arxiv: 1803.10082 (2018).
43. Han, S., Pool, J., Tran, J. & Dally, W. Learning both weights and connections for efficient neural network. *Adv. Neural Inf. Process. Syst.***28** (2015).
44. Frankle, J. & Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations* (2019).
45. Tartaglione, E., Lepsøy, S., Fiandrotti, A. & Francini, G. Learning sparse neural networks via sensitivity-driven regularization. *Adv. Neural Inf. Process. Syst.***31** (2018).
46. Sanh, V., Wolf, T. & Rush, A. Movement pruning: Adaptive sparsity by fine-tuning. *Adv. Neural. Inf. Process. Syst.* **33**, 20378–20389 (2020).
47. Kohama, H., Minoura, H., Hirakawa, T., Yamashita, T. & Fujiyoshi, H. Single-shot pruning for pre-trained models: Rethinking the importance of magnitude pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1433–1442 (2023).
48. Gupta, M. et al. Is complexity required for neural network pruning? a case study on global magnitude pruning. In *2024 IEEE Conference on Artificial Intelligence (CAI)*, pp. 747–754 (IEEE, 2024).
49. Hoffer, E., Banner, R., Golan, I. & Soudry, D. Norm matters: efficient and accurate normalization schemes in deep networks. *Adv. Neural Inf. Process. Syst.***31** (2018).
50. Zhang, C., Bengio, S. & Singer, Y. Are all layers created equal?. *J. Mach. Learn. Res.* **23**, 1–28 (2022).
51. Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256 (JMLR Workshop and Conference Proceedings, 2010).
52. He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference On Computer Vision*, pp. 1026–1034 (2015).
53. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (2016).
54. Lee, N., Ajanthan, T. & Torr, P. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations* (2019).
55. Tartaglione, E., Bragagnolo, A., Fiandrotti, A. & Grangetto, M. Loss-based sensitivity regularization: towards deep sparse neural networks. *Neural Netw.* **146**, 230–237 (2022).
56. LeCun, Y., Denker, J. & Solla, S. Optimal brain damage. *Adv. Neural Inf. Process. Syst.***2** (1989).
57. Yao, Z., Gholami, A., Keutzer, K. & Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE International Conference on Big Data (Big data)*, 581–590 (IEEE, 2020).
58. Yu, S. et al. Hessian-aware pruning and optimal neural implant. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 3880–3891 (2022).
59. Yang, H. et al. Global vision transformer pruning with Hessian-aware saliency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18547–18557 (2023).
60. Pessia, R., Ishigami, P. G. & Jodelet, Q. Artificial lunar landscape dataset (2019).
61. Xiao, X., Yao, M. & Liu, H. Marsdata-v2, a rock segmentation dataset of real martian scenes (2022).
62. Ronneberger, O., Fischer, P. & Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*. arxiv: 1505.04597 (2015).
63. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. *CoRR*. arxiv: 1512.03385 (2015).
64. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints* (2014). arxiv: 1409.1556.
65. Tan, M. & Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*. arxiv: 1905.11946 (2019).
66. Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A. & Chen, L. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*. arxiv: 1801.04381 (2018).

67. Vasu, P., Gabriel, J., Zhu, J., Tuzel, O. & Ranjan, A. Mobileone: An improved one millisecond mobile backbone. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7907–7917, https://doi.org/10.1109/CVPR52729.2023.00764 (IEEE Computer Society, Los Alamitos, CA, USA, 2023).
68. Huang, G., Liu, Z. & Weinberger, K. Q. Densely connected convolutional networks. *CoRR*. arxiv: 1608.06993 (2016).
69. Xie, E. et al. Segformer: simple and efficient design for semantic segmentation with transformers. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21 (Curran Associates Inc., Red Hook, NY, USA, 2024).
70. Kosmidis, L. et al. Gpu4s: Embedded gpus in space - latest project updates. *Microprocessors and Microsystems***77** (2020).

Acknowledgements

This paper has been supported by the French National Research Agency (ANR) in the framework of the JCJC project "BANERA" ANR-24-CE23-4369, and by Hi!PARIS Center on Data Analytics and Artificial Intelligence.

Author contributions

E.T. conceived the experiment(s), L.O. and E.S.M. conducted the experiment(s), L.O. and E.S.M. analyzed the results. All authors reviewed the manuscript.

Declarations

Competing interests

The corresponding author is responsible for submitting a competing interests statement on behalf of all authors of the paper.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-025-99192-5.

Correspondence and requests for materials should be addressed to E.T.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.