



**TUM Data Innovation Lab**  
Munich Data Science Institute (MDSI)  
Technical University of Munich

&

**TUM Chair of Data Science in Earth  
Observation (MDSI Prof. Xiaoxiang Zhu)**  
cooperating with the  
**German Aerospace Center (DLR)**

Final report of project:

**Towards a NAS Benchmark for Classification  
in Earth Observation**

Authors      B.Sc. Pavel Jahoda, B.Sc. Safa Yilmaz, B.Sc. Emre Demir,  
                  B.Sc. Viet Nguyen, B.Sc. Zhihang Liu,  
Mentor(s)    Dr. Andrés Camero, M.Sc. Kalifou René Traoré, M.Sc. Kon-  
                  stantin Riedl  
Project Lead  Dr. Ricardo Acevedo Cabra (MDSI)  
Supervisor   Prof. Dr. Massimo Fornasier (MDSI)

February, 2023

## Abstract

While classical computer vision algorithms achieve remarkable success in several real-world applications, they are not applicable to the Earth Observation (EO) domain due to the inherent complexity of satellite data. For this reason, new and more tailored architectures are required to effectively represent EO data. In this project, we aim at creating the first-ever NAS database specifically for the classification tasks of EO data. This is done by leveraging the search space of NASBench-101, where a collection of architectures was evaluated using the CIFAR-10 dataset for image classification tasks. Instead of exhaustively searching and evaluating each model, we employ surrogate benchmarking on this search space. A subset of architectures is evaluated on our own satellite dataset So2Sat LCZ42 that is 8 times larger compared to CIFAR-10. The quality of our neural network architecture database is assessed by using Fitness Landscape Analysis (FLA), which provides insights into the difficulty of the architecture search problem by measuring the performance of models over time as well as the consistency of the search trajectories. We utilize the JUWELS supercomputer and implement distributed training with the Distributed-DataParallel (DDP) method for PyTorch, which enables multiple GPUs to work together per node. This helps to reduce the training time by 80% to approximately 2.5 hours per architecture, allowing the search and evaluation of 907 architectures in the search space within the given computational budget. The results of our analysis show that the So2SAT LCZ42 dataset presents a more challenging landscape compared to CIFAR-10.

# Contents

<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Related Work</b>	<b>3</b>
2.1 Neural Architecture Search . . . . .	3
2.1.1 Search Space . . . . .	4
2.1.2 Search Strategy . . . . .	5
2.1.3 Performance Estimation Strategy . . . . .	6
2.2 NAS-Bench-101 . . . . .	6
2.3 NASLib . . . . .	6
2.3.1 Surrogate NAS Benchmarks . . . . .	7
2.4 Landscape Analysis . . . . .	8
2.4.1 Fitness Landscape Footprint . . . . .	8
2.4.2 Landscape of Neural Architecture Search Across Sensors . . . . .	9
<b>3 Method</b>	<b>9</b>
3.1 Surrogate benchmark and search space . . . . .	9
3.2 Accessing the database . . . . .	10
<b>4 Experiments</b>	<b>11</b>
4.1 Data Description . . . . .	11
4.1.1 Encoding Scheme . . . . .	15
4.2 Experimental Settings . . . . .	15
4.2.1 JUWELS . . . . .	16
4.2.2 Training pipeline . . . . .	17
4.3 Training Setting . . . . .	20
<b>5 Results</b>	<b>21</b>
<b>6 Conclusion</b>	<b>22</b>
<b>Appendix</b>	<b>24</b>

# 1 Introduction

The success of deep learning is largely driven by the creation of advanced neural network architectures. However, the manual design is a cost-intensive process that requires expert knowledge in the field. This has led to the invention of Neural Architecture Search (NAS), a technique for automating the design of high-performance neural network architectures. NAS reduces the cost of implementing neural networks and enables robust representation learning on a given dataset. NAS methods have achieved state-of-the-art results in various computer domains, including semantic segmentation[10], image classification [17], transfer learning, machine translation and reinforcement learning [5]. Furthermore, they have even surpassed human-designed architectures [24]. For instance, NASBench-101 [31], a public architecture dataset for NAS research, is based on the CIFAR-10 dataset and has enabled NAS researchers to reduce the cost of inferences of search algorithms via simple cost-free queries to the database. However, this benchmark has limitations, including a limited search space and the time-consuming process of exhaustive search. In contradiction, NAS has recently seen a progressive increase in the search space size by several orders of magnitude [35, 13, 11]. To address the computational challenges faced when exploring the search space, we leverage the principles of surrogate NAS benchmarks [21], which allows to significantly reduce the requirements on computational resources. Surrogate benchmarking is a technique that predicts the performance of neural architectures by extrapolating the accuracy performance evaluated on a subset of models in the search space, opposed to undergoing an exhaustive search process. Current state-of-the-art NAS surrogate algorithms and benchmarks are evaluated on CIFAR-10 dataset [35, 13, 31, 21] of images, where some approaches already surpassed 99.5% accuracy[4, 1]. This is due to the fact that this classification dataset has limited complexity. By providing a surrogate NAS database for the task of classifying images, we advance the realm of NAS to intrinsically more complex dataset, So2Sat LCZ42, in EO domain. To the best of our knowledge, this is the first time an EO dataset is used in a surrogate NAS setting. Furthermore, we propose a methodology for controlling the quality of a surrogate NAS benchmark based on a fitness landscape analysis tool called fitness landscape fitness footprints.

In Sec. 2, we revisit Neural Architecture Search, the surrogate setting, and landscape analysis. In addition, we summarize state-of-the-art methods in this domain. This is followed by Sec. 3, where we describe the dataset So2Sat LCZ42, and the methodology of our work. We then provide an analysis of experiment’s results together with a statistical analysis of the surrogate NAS benchmark in Sec. 5. This is followed by Sec. 6 where we not only discuss the results but also give an outline for future work.

## 2 Related Work

### 2.1 Neural Architecture Search

Neural Architecture Search (NAS) is a subfield of Automated machine learning (AutoML), which is concerned with the process of automating the tasks of applying machine learning to real-world problems. AutoML potentially includes every stage from beginning with a raw dataset to building a machine learning model ready for deployment. In contrast, NAS

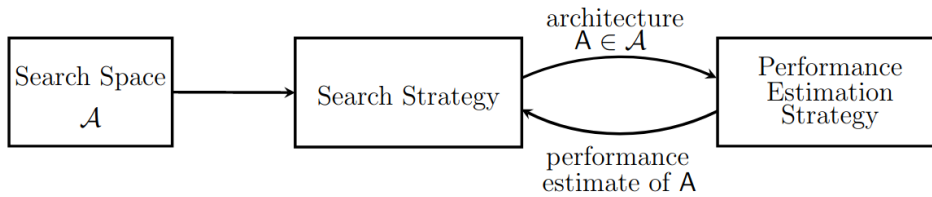


Figure 1: Abstract illustration of Neural Architecture Search process, adopted from [5]

focuses on how to automate the process of finding neural network architecture by optimizing the topology of the network. NAS methods have outperformed manually designed architectures on several tasks such as image classification [34, 35, 17], object detection [35] or semantic segmentation [2]. NAS methods typically define a set of “building blocks” and rules on how to connect them. Then they devise a search strategy that selects an architecture  $\mathbf{A}$  from a predefined search space  $\mathcal{A}$ , which is passed to a performance estimation strategy. The returned estimated performance of  $\mathbf{A}$  to the search strategy is used to refine the architecture. This process is depicted in Fig. 1. In the next sections, we review search space, search strategy, and estimation strategy advancements in recent years.

### 2.1.1 Search Space

A search space refers to the set of possible solutions for the neural network architectures. It encompasses all the different hyperparameters, architectural choices, and other factors that determine the final design of the network. The search space is considered a conditional space because the options for layer hyperparameters, for instance, are limited by the type of operation the layer performs. This means that the choices for these hyperparameters depend on the specific type of operation being executed. One of the most seminal works [35] looks at the search space in terms of cell-based representation, which contains two different cells. The first one is called a normal cell that does not change the dimensionality of the input as opposed to the reduction cell. A cell contains several operations predicted by the NAS algorithm (e.g. multiple convolution layers). In the cell-based representation, a neural architecture is defined as the same cell repeated multiple times. Limiting the search space in such a way allows to speed up the NAS process and scalability as the number of cells can be defined depending on the available computational resources. In this type of search space, micro-architecture optimization refers to optimizing a single cell, while macro-architecture optimization refers to optimizing the entire architecture.

For design principles, macro-architectures and micro-architectures should be jointly optimized. If only micro-architectures are optimized, manual macro-architecture engineering needs to be performed after finding the optimal cells. A remedy to optimize macro-architectures is to design a hierarchical search space [12] that consists of multiple levels. The first level consists of a set of basic building blocks (primitive operations). The second level involves different combinations of these primitive operations (motifs) that are connected to each other in a directed acycle graph. The third level involves higher-level combinations of these second-level motifs. A caveat to the search spaces is they are manually designed, which might contain human biases and constraint the philosophy of

AutoML. The choice of the search space depends mainly on the difficulty of the optimization problem that remains non-convex, non-continuous and high-dimensional.

### 2.1.2 Search Strategy

The search strategy details how to explore the search space (which is often exponentially large or even unbounded). It encompasses the classical exploration-exploitation trade-off of metaheuristics since, on the one hand, it is desirable to find well-performing architectures quickly, while on the other hand, premature convergence to a region of sub-optimal architectures should be avoided. Many different search strategies can be used to explore the space of neural architectures, including random search, evolutionary methods, reinforcement learning (RL), Bayesian optimization, and gradient-based methods. Although the foregoing methods have been actively used by researchers for different AI-related tasks, NAS could gain momentum in the machine learning community after the work conducted by Zoph and Le [34] that the found models achieved better performance on benchmark datasets compared to that of manually designed models. Despite the heavy computational requirement of the work, many methods have been published to ease this problem.

NAS can be viewed as an RL problem, with the generation of a neural architecture as the agent’s action and the search space as the action space. The reward is based on the performance of the trained architecture during inference. Different RL techniques differ in how the agent’s policy is designed and optimized. For example, Zoph and Le [34] use a recurrent neural network policy and train it with the REINFORCE policy gradient algorithm. Another approach to NAS are neuro-evolutionary [5] methods that use evolutionary algorithms to optimize the architecture. However, due to the scalability of large neural networks, gradient-based weight optimization methods currently perform better than evolutionary methods. Many recent neuro-evolutionary techniques use gradient-based methods for weight optimization and evolutionary algorithms for architecture optimization. In neuro-evolutionary methods, a population of trained networks is evolved. At each step, one or more models from the population are selected as parent networks and offspring are generated by mutating the parent. After the child models are trained, their performance is tested, and they are added to the population. Neuro-evolutionary methods vary in parent selection techniques, population updates, and offspring generation methods. Parent selection methods may include sampling from a multi-objective Pareto front and tournament selection, while population updates may include removing the worst individuals or not removing any. Offspring can be generated by initializing the child network randomly or by using different inheritance methods.

Bayesian optimization (BO) [20] is a method used for hyperparameter optimization. The technique is based on Gaussian processes and is used for optimizing low-dimensional continuous problems. For example, Swersky et al. design a kernel function for the search space to apply BO methods for NAS [23]. Evidence suggests that BO performs better than evolutionary algorithms. However, all of the techniques discussed so far are designed for discrete search spaces where only weights can be optimized using gradient-based methods. Liu et al. [13] proposed DARTS that can optimize both the architecture and weights end-to-end by using a continuous relaxation for the search space. This method alternates

between optimizing the weights using gradient descent on training data and optimizing the architecture using gradient descent on validation data before it determines the best operation for each layer by taking the argmax. Some methods have also been proposed to optimize the parametrized distribution [29] over operations to better measure the uncertainty of the networks, rather than just optimizing the coefficients.

### 2.1.3 Performance Estimation Strategy

The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data. Performance Estimation refers to the process of estimating this performance: the simplest option is to perform standard training and validation of the architecture on data, but this is unfortunately computationally expensive and limits the number of architectures that can be explored. Much recent research, therefore, focuses on developing methods that reduce the cost of these performance estimations [14, 18].

There are four main focuses on how to speed up the NAS algorithms. The first direction is to lower the fidelities of model performances after training. These indicators include shorter training time, training on a subset of a dataset, on lower-resolution images, or with fewer filters per layer or fewer cells. The second method to speed up is to utilize learning curve extrapolation to early terminate candidates that perform poorly. This helps to boost the search process. Based on this idea, Liu et al. [12] proposed a surrogate model that supports predicting performance based on cell or architectural properties and extrapolates to those with larger sizes during training. The third approach is to initialize the weights of novel architectures based on those of the models that have been trained before (weight inheritance or network morphisms). The final method is a one-shot architecture search, where all architectures are treated as different subgraphs of a supergraph (one-shot model) and share weights. By doing this, only the one-shot model needs to be trained and the sub-architectures can be separately evaluated during inference time.

## 2.2 NAS-Bench-101

NASBench-101 [31] was designed to address the challenges faced in the evaluation of Neural Architecture Search (NAS) algorithms. These evaluations were previously time-consuming and required a lot of resources, making it difficult to progress in the field of NAS research. NASBench-101 provides a cost-effective, fast, and scalable solution for evaluating the performance of NAS algorithms, including optimization of architecture and hyperparameters. It offers a set of pre-trained neural network architectures for image classification tasks on the CIFAR-10 [9] dataset, along with metrics like training time and accuracy on train, validation, and test splits, making it possible to compare the performance of different NAS algorithms.

## 2.3 NASLib

NASLib [19] is a modular and flexible NAS library that helps researchers to lightly implement and extend state-of-the-art NAS methods, and has been used in many NAS studies [8, 28]. With a few lines of code, researchers can easily reuse search spaces, benchmark interfaces, and evaluation pipelines, including the NASBench-101 benchmark used in this



study.

NASLib consists of 4 main building blocks that can interact with each other[19]:

- Search spaces (cell search space, hierarchical)
- Optimizers (one-shot/weight-sharing optimizers, black-box optimizers)
- Predictors (estimators that, given an architecture as input, output its performance)
- Evaluators (run the architecture search loop and the final network training pipeline)

The search space is the most critical part of NASLib since the best hyperparameters depend on the specific search space. NASLib conducts NAS space searches mainly in the form of directed acyclic graphs (DAGs), and it inherits NetworkX’s basic graph classes to create and manipulate graphs, where the properties of nodes and edges can be arbitrary Python objects, allowing us to quickly implement different components (e.g., cells in a macro architecture, graphs nested at multiple levels)[19].

NASLib optimizers interact with the search space and search for the best architecture, mainly including one-shot/weight-sharing optimizers and black-box optimizers. With the 31 performance predictors provided by NASLib, the evaluation of specific benchmarks. In summary, using NASLib, we can quickly and comprehensively evaluate new NAS algorithms on Earth Observation problems and improve our research’s experimental rigour and generality based on some of the benchmark suites it provides.

### 2.3.1 Surrogate NAS Benchmarks

In recent years, the search spaces of NAS algorithms have progressively increased in size from PNAS [11], DARTS [13], which contain more than  $10^{14}$  and  $10^{18}$  architectures respectively, up to search spaces that contain more than  $10^{28}$  architectures [35]. For these search spaces, exhaustive search of all possible architectures is infeasible. Overcoming this issue is possible by using a surrogate NAS benchmark [32].

To foster reproducibility and accessibility, Siems et al. developed **NASBench-301** [21] that covers the cell-based search space of DARTS containing more than  $10^{18}$  possible architectures. The benchmark is proposed for evaluating surrogate functions for architecture search instead of performing exhaustive computation for the entire collection of all architectures in the search space. The NASBench-301 was created by first, full training 60k unique architectures on the CIFAR-10 dataset. Then, the authors trained a difference performance predictor model including a graph convolutional network called GIN [30]. The predictors are trained on the classification accuracies of the 60k architectures which can then be used to estimate the accuracy of the remaining  $10^{18}$  possible architectures.

Due to the “locality” property of the models in the search space, the accuracy statistics of a model can explain its neighborhood very well. Therefore, a surrogate (predictive) model trained on the dataset can outperform the tabular method. The surrogate technique is very useful for building new benchmarks for architecture search by being able to operate

in large spaces and reduce computational resources vastly. For our project in Landscape analysis, we will focus on constructing our architecture dataset following this methodology.

## 2.4 Landscape Analysis

Fitness Landscape Analysis (FLA) aims to characterize optimization problems [16] The fitness landscape is defined as the triplet combination  $z = (S, f, N)$  where  $S$  denotes the search space of all the possible solutions, in our case NN architectures, by a fitness function  $f$  that assigns value to each candidate from the search space (for example classification accuracy) and a neighborhood operator  $N$  to navigate  $S$ . In our case, we can represent each neural network architecture as a binary vector and subsequently define neighborhood operator  $N$  as a hamming distance and say that the neighborhood of an architecture is a set of architectures that have a hamming distance from the architecture. It can be seen that FLA aims at describing the hardness of a problem, using limited knowledge of an algorithm to be used.

### 2.4.1 Fitness Landscape Footprint

The fitness landscape footprint is an aggregation of eight general-purpose metrics to quantitatively describe such landscape [27].

1. **Mean fitness:** The mean classification accuracy of all the model architectures.
2. **Variance fitness:** The variance of classification accuracies.
3. **Ruggedness:** A metric measured by autocorrelation  $\rho$  of fitness of random walks (hamming distance 1) in a search space. To get a single value, we use a mean function.
4. **Cardinal of optima:** An estimate of the number of local optima in  $S$ .
5. **Persistence:** The probability that model keeps a rank (a position in terms of classification accuracy compared to other models) it had after  $t_0$  iteration when trained after  $k$  iteration. Then, we define persistence  $\Pi$  as the probability for model configurations ranked by function  $\text{Ranking}(\cdot)$  to keep their initial rank (i.e., the one observed at  $t_0$ ) through  $R_{all}$ .
6. **Positive persistence:** The probability that the best models remain best performing.
7. **Negative persistence:** The probability that the worst performing models remain worst performing.
8. **Persistence positive AuC:** Area under the Curve of positive persistence.
9. **Persistence negative AuC:** Area under the Curve of negative persistence.

In the paper, the authors used the footprint to describe the landscape of the NasBench-101 benchmark dataset and the So2Sat LCZ42 dataset. The NasBench-101 is a benchmark for NAS methodologies in the context of image classification. It consists of 453k distinct

CNN architectures and their fitness evaluation at various training steps (4, 12, 36, and 108 epochs), on the dataset of CIFAR-10. In the following text, two main findings are highlighted. First, with approximately 75% more local optima on So2Sat LCZ42 (11153 versus 6373), the chances of being trapped in a non-optimal region of the space are higher on So2Sat LCZ42 than on CIFAR-10. Then, solution diversity should be considered when designing a (local search-based) algorithm to do NAS on So2Sat LCZ42 (i.e., exploration-exploitation). Second, elite models persist over time, i.e., the chances of finding a model that will be a top 25% performer in a test from 4 to 108 epochs of training are 32%. Moreover, on So2Sat LCZ42, elite models (rank < 25%) will remain on the top with a higher probability (0.48). Therefore, spotting elite performers early could help to improve NAS performance. However, overall, they observe similar characteristics in both footprints. Therefore, we may extrapolate from one problem to the other.

#### 2.4.2 Landscape of Neural Architecture Search Across Sensors

In the study done by Traoré et al., the authors leverage the footprint methodology to compare how much the fitness landscape differs across sensors [25] (specifically with respect to sensor data from satellites Sentinel-1, Sentinel-2, and a fusion of the sensors). The paper considers a case of a fixed search space, training pipeline (hyperparameters, duration, etc.), and evaluation protocol (fitness function). The fixed search space, i.e., the search space of architectures to be evaluated, can be used, because for the fusion of the sensors, the sensors are fused by simply stacking (addition) the data at the input level, so the NN architectures don't need to be modified. Similarly, as in the study introducing Fitness Landscape Footprint [27], the landscape is analyzed on the NASBench-101 dataset representing each neural network architecture as a binary vector. In terms of probability density functions of classification accuracies of different architecture, they find that training on Sentinel-2 ( $\mu = 0.94$ ,  $\sigma^2 = 0.03$ ) outperforms training on Sentinel-1 ( $\mu = 0.47$ ,  $\sigma^2 = 0.13$ ) as well as fusing both sensors ( $\mu = 0.89$ ,  $\sigma^2 = 0.05$ ). It is surprising that with the current search space, the search behavior is worse when using both sensors as input. However, not that only one fusion configuration was considered.

## 3 Method

Our methodology for the project involves using the So2Sat LCZ24 dataset [33] for searching, evaluating neural architectures and applying surrogate benchmark on the NASBench-101 search space, given time constraints and computational resources. We search and evaluate a subset of architectures in the search space using the So2Sat LCZ24 dataset, then fit a surrogate model on their validation accuracies in order to predict the performance of the remaining models in the space. To reduce the training complexity, we employ the distributed data parallel technique to train the architectures on multiple GPUs. The quality of the neural architecture database is assessed by the landscape analysis footprints.

### 3.1 Surrogate benchmark and search space

Due to compatibility issues with NASBench-301 in NASLib, we use NASBench-101 as our search space since it is readily available and we were able to create an architecture

database using the So2Sat LCZ42. Our technical choices are as follows:

- **Search Space.** We employ NASBench-101 search space for searching and evaluating architectures on the So2Sat LCZ42 dataset [33].
- **Evaluation Metrics.** We use the footprints described in Sec. 2.4.1 to compare the final evaluated architectures in the search space. For the training process, we keep track of the average macro accuracy, average micro accuracy, and per-class accuracies of every model. While the micro average keeps track of average per-class accuracy, the macro average can potentially reflect the true model performance since we have a very imbalanced dataset, where some classes contain very fewer data points compared to other major classes.
- **Surrogate Model.** We had consumed all of our compute hours for doing random walk on the sampled architectures, training of surrogate model is not feasible. Thus, we use a surrogate model on the NASBench-101 search space to predict the performance of candidate architectures without having to train them completely. This involves training on a subset of evaluated architectures and using their validation accuracy to fit the surrogate model. We currently employ a Gradient Boosting Machine (GBM), which is an ensemble model of multiple decision trees. Specifically, we use an open-source implementation of GBM provided by the XGBoost library [3].

### 3.2 Accessing the database

- **Binary Feature Representation.**

Each architecture in NASBench-101 [31] is identified by two parameters, a 7x7 upper-triangular module adjacency and an array with a length of 6 to represent the module operations. The encoding strategy is based on these two parameters to uniquely encode all 423 624 architectures in the search space. Each architecture is represented by a binary-encoded value with a length of 289. Thus, a hamming distance can be defined between any two architectures to move in the search space.

- **Sampling, Random Walk, and Local Estimations (FLA).**

Due to the size of the search space, a reasonable strategy to evaluate the performance of the architectures is required to be decided. An illustration of the employed strategy, namely the Design of Experiment (DOE), can be seen in Fig. 2. Every point in the search space of our experiment represents an architecture from NASBench-101 which has 423 624 architectures in total. Identifying the performance of the search space on So2Sat LCZ42 dataset is initialized by choosing starting points in the space. This phase is completed by randomly choosing architectures from the space, then performances of these points are evaluated Fig. 2(a). In the coming step, to enlarge the examined field, we randomly choose from sampled points and walk to their neighbors. This operation is performed multiple times to expand the examined area as much as possible. During this operation, we need to navigate inside the space, and it is achieved by employing our neighborhood operator which basically changes 1-bit of given encoded input, thus finds a point which is 1-Hamming distance away from the starting point, Fig. 2(b). At the end of second step, 907 architectures in

the space are evaluated and it took approximately 120.000 compute hours. At the last stage of experiment, we aim to move further from these accumulated points by randomly choosing from them and using a similar strategy as in the second step. In other words, we intend to visit new neighbors which are 1-hamming distance away from the new chosen point. This stage is called local search Fig. 2(c). Considering the required time to train 1 architecture, approximately 2.54 hours, with the complexity of dataset, this stage of the experiment is standing as a next step for further studies. However, gathered results from the 907 architectures already gave a promising result to understand the performance search space on our dataset.

## 4 Experiments

### 4.1 Data Description

Our surrogate benchmark uses data from the So2Sat LCZ42 dataset [33] to evaluate our NAS search space. For large or global scale remote sensing areas, generating land use/land cover (LULC) maps play a critical role to clarify some generic concepts including, climate change, sustainable urban development or even providing information for disaster management. Among the mentioned concepts, urbanization has the dominant share and LULC maps provide useful information to follow and manage urban growth. Although there exist some useful LULC materials obtained by using remote sensing, they all provide semantic labels as urban or non-urban, in some cases finer classes. However, labels presented in those studies can be debatable since mentioned semantic labels might depend on human interpretation and are culture-specific. For instance, urban and non-urban categorization can be defined differently in Europe and Africa and also can be classified differently by individuals. At this point, local climate zone (LCZ) plays a key role to represent an objective and culture-independent classification methodology.

LCZs are composed of 10 built classes and 7 natural ones. They are called local climate zones since classes are based on climate-relevant surface properties. For example, the height and density of buildings as surface structures and trees, vegetation or palings as surface cover, and anthropogenic parameters such as human-based heat output. A representative illustration of LCZ classes can be seen in Fig. 3. Due to the nature of LCZ

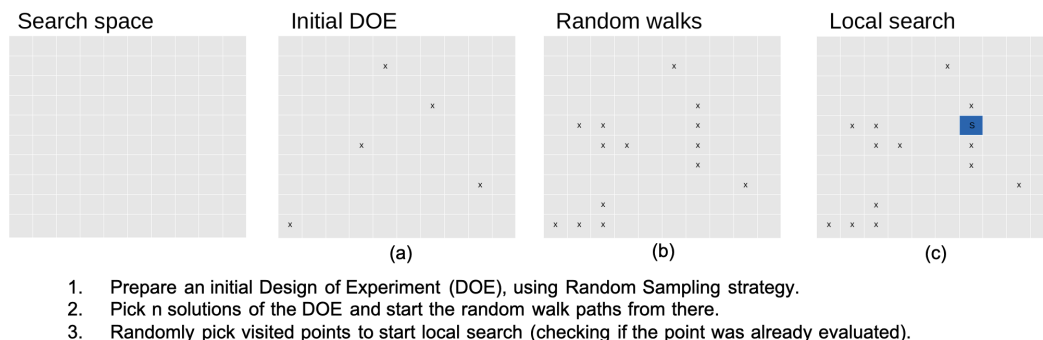


Figure 2: Sampling strategy

classes that are defined according to the physical properties of obtained data, they can be applied to the different regions on the earth. This classification method also brings an opportunity for researchers to work on different subjects such as infrastructure planning, disaster mitigation, and city planning.

To be able to provide such a database, 400,673 pairs of Sentinel-1 and Sentinel-2 images were labeled. The data were obtained from two different satellites called Sentinel-1 and Sentinel-2. Sentinel-1 is the first of the Copernicus Programme satellite constellation conducted by the European Space Agency. The Sentinel-1 performs C-band synthetic aperture radar (SAR) imaging. It uses the motion of the radar antenna over a target region to provide fine spatial resolution resulting in two-dimensional images that describe the earth's landscape. The data from Sentinel-1 consist of 8 spectral real-valued bands. On the other hand, Sentinel-2 uses MultiSpectral Instrument (MSI) to collect data from 13 spectral bands. Out of which, 10 are used in the So2Sat LCZ42 dataset with central wavelengths of these bands ranging from  $0.49\ \mu\text{m}$  to  $2.19\ \mu\text{m}$ . Images from both satellites have been pre-processed to have  $32 \times 32$  dimension, where each pixel corresponds to  $10 \times 10$  meters on the ground. Sample data from Sentinel-1 and Sentinel-2 in the LCZ42 So2Sat dataset can be seen depicted in Fig. 4.

These images were selected from different continents apart from Antarctica. Samples include 52 different regions, and only 10 of them represent relatively smaller areas. To achieve a high-quality database, a well-defined workflow and decision rules were employed. Before the labeling procedure, a pre-meeting was conducted to ensure that the understanding of the 17 classes have the same meaning for all the members of the labeling crew. After the pre-labeling step, results were inspected by different members to identify potential errors. In the final step, a quantitative evaluation of the label quality was decided.

The labeling procedure is composed of four main steps, learning, labeling, visual validation, and quantitative validation. Although LCZs are less subjective compared to the existing classification methods, as mentioned, a learning process is still required since an understanding of the 17 classes are needed to be clarified. Moreover, some areas may not fall into any of the LCZ classes or some of them can fall into multiple classes. Members of the crew were enabled to understand the 17 classes using their representative samples from Google Earth. As a first step during labeling, a couple of cities were chosen by the labeling members, and assigned cities were further clarified by drawing a 50x50 kilometers rectangle. After this step, specified regions were delineated in Google Earth and used as preliminary labels. As a last part of this initial step, Landsat 8 images of these specified regions in the selected cities were created.

In the upcoming stage, using those labels and Landsat 8 images a random forest classifier was trained to produce an LCZ classification map for the desired city. This map was used to provide a guide to label images; at the end of the procedure, the provided data set includes only human-labeled images. This classification map and the images on Google Earth were employed to check the correctness and completeness of the LCZ labels. The correctness of the obtained result was strengthened by inspecting the differences between the classification map and the manual labeling of members. In case of a difference; an-

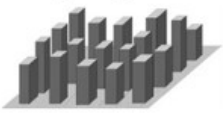
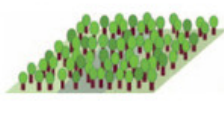



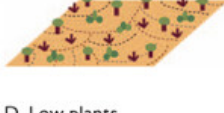



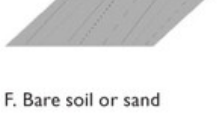
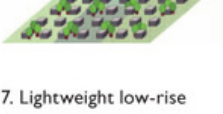
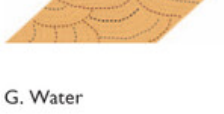




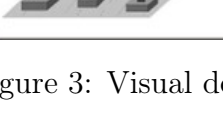
Built types	Definition	Land cover types	Definition
 <p>1. Compact high-rise</p>	Dense mix of tall buildings to tens of stories. Few or no trees. Land cover mostly paved. Concrete, steel, stone, and glass construction materials.	 <p>A. Dense trees</p>	Heavily wooded landscape of deciduous and/or evergreen trees. Land cover mostly pervious (low plants). Zone function is natural forest, tree cultivation, or urban park.
 <p>2. Compact midrise</p>	Dense mix of midrise buildings (3–9 stories). Few or no trees. Land cover mostly paved. Stone, brick, tile, and concrete construction materials.	 <p>B. Scattered trees</p>	Lightly wooded landscape of deciduous and/or evergreen trees. Land cover mostly pervious (low plants). Zone function is natural forest, tree cultivation, or urban park.
 <p>3. Compact low-rise</p>	Dense mix of low-rise buildings (1–3 stories). Few or no trees. Land cover mostly paved. Stone, brick, tile, and concrete construction materials.	 <p>C. Bush, scrub</p>	Open arrangement of bushes, shrubs, and short, woody trees. Land cover mostly pervious (bare soil or sand). Zone function is natural scrubland or agriculture.
 <p>4. Open high-rise</p>	Open arrangement of tall buildings to tens of stories. Abundance of pervious land cover (low plants, scattered trees). Concrete, steel, stone, and glass construction materials.	 <p>D. Low plants</p>	Featureless landscape of grass or herbaceous plants/crops. Few or no trees. Zone function is natural grassland, agriculture, or urban park.
 <p>5. Open midrise</p>	Open arrangement of midrise buildings (3–9 stories). Abundance of pervious land cover (low plants, scattered trees). Concrete, steel, stone, and glass construction materials.	 <p>E. Bare rock or paved</p>	Featureless landscape of rock or paved cover. Few or no trees or plants. Zone function is natural desert (rock) or urban transportation.
 <p>6. Open low-rise</p>	Open arrangement of low-rise buildings (1–3 stories). Abundance of pervious land cover (low plants, scattered trees). Wood, brick, stone, tile, and concrete construction materials.	 <p>F. Bare soil or sand</p>	Featureless landscape of soil or sand cover. Few or no trees or plants. Zone function is natural desert or agriculture.
 <p>7. Lightweight low-rise</p>	Dense mix of single-story buildings. Few or no trees. Land cover mostly hard-packed. Lightweight construction materials (e.g., wood, thatch, corrugated metal).	 <p>G. Water</p>	Large, open water bodies such as seas and lakes, or small bodies such as rivers, reservoirs, and lagoons.
 <p>8. Large low-rise</p>	Open arrangement of large low-rise buildings (1–3 stories). Few or no trees. Land cover mostly paved. Steel, concrete, metal, and stone construction materials.	<b>VARIABLE LAND COVER PROPERTIES</b>	
 <p>9. Sparsely built</p>	Sparse arrangement of small or medium-sized buildings in a natural setting. Abundance of pervious land cover (low plants, scattered trees).	<p><i>b. bare trees</i></p>	Leafless deciduous trees (e.g., winter). Increased sky view factor. Reduced albedo.
 <p>10. Heavy industry</p>	Low-rise and midrise industrial structures (towers, tanks, stacks). Few or no trees. Land cover mostly paved or hard-packed. Metal, steel, and concrete construction materials.	<p><i>s. snow cover</i></p>	Snow cover >10 cm in depth. Low admittance. High albedo.
		<p><i>d. dry ground</i></p>	Parched soil. Low admittance. Large Bowen ratio. Increased albedo.
		<p><i>w. wet ground</i></p>	Waterlogged soil. High admittance. Small Bowen ratio. Reduced albedo.

Figure 3: Visual description of the 17 LCZ42 So2Sat dataset classes, adopted from [22]



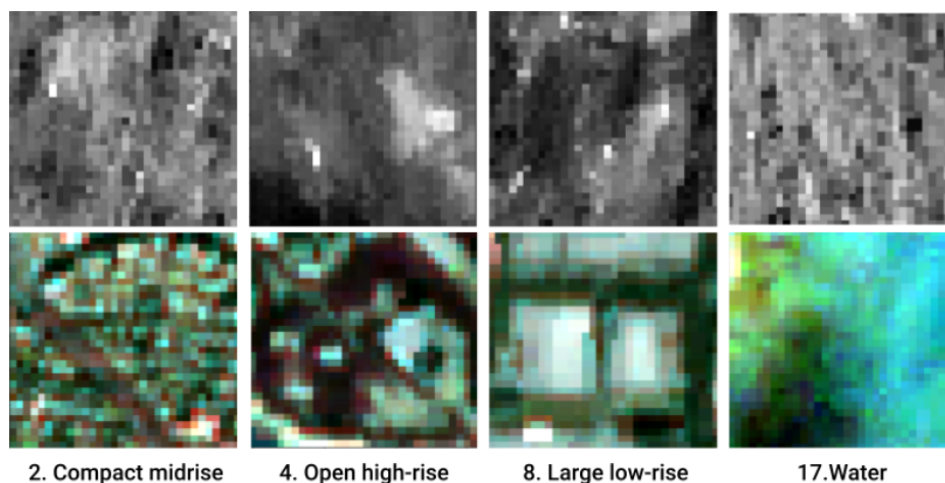


Figure 4: So2Sat LCZ42 samples. Examples of Sentinel-1 SAR patches (**top row**), followed by the corresponding Multi-spectral Sentinel-2 patches (**bottom row**), adopted from [25]

other manual re-correction is performed by checking the related areas on Google Earth. Another criterion – completeness is performed by deciding unlabeled areas again by using Google Earth, to find negative samples.

After this manual labeling procedure, to get rid of personal bias and outliers, another manual inspection process was conducted. This stage was performed by two persons who did not label the current interested area. Two different types of inspections were performed at this stage. The first one is called an obvious outlier, such as water being classified as a dense-high-rise building. In the case of this problem, a polygon with the correct label was added. The latter problem, a normal compactness-centric pattern, required more inspection to correct since such a problem is more difficult to identify since it represents more complex problems such as the compactness of urban buildings decreasing from the city to suburban. To solve this problem, adding more polygons or correcting the labels of existing polygons are required.

As a last step, obtained results were fined to get better results. This stage includes examining close pixels of different LCZ classes. To overcome this issue, shrinking the polygon of all non-urban LCZ classes apart from the water was conducted. Another improvement stage is to balance the classes; since non-urban LCZ classes are naturally much larger than the urban ones, non-urban classes were reduced to the maximum number of samples from urban classes. Since it is difficult to label urban classes, they kept them as they were.

Finally, the obtained dataset reached a volume of 56GB, including 400,673 samples. The total data set was split into 3 categories: 352,366 training samples, 24,119 test samples, and 24,119 validation samples. As an important point, it is underlined that the test set and training set are composed of different regions which increases the reliability of the dataset. Trained models with this dataset including, random forest, support vector machines, and attention-based ResNeXt, and overall accuracy obtained between 0.51 and 0.61. It



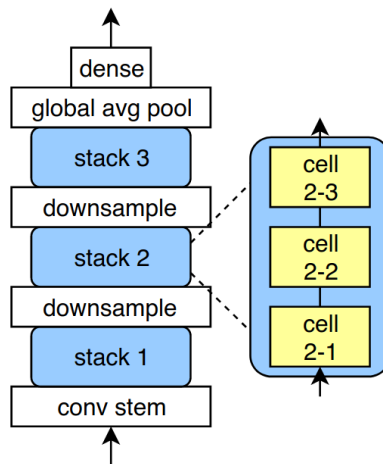


Figure 5: Illustration of NASBench-101 neural network architecture design used to define the search space, adopted from [31]

is emphasized that overall accuracy between 0.5 and 0.6 can be considered promising; however, to satisfy the land cover mapping purpose, a value of 0.85-0.9 is still required.

#### 4.1.1 Encoding Scheme

Our surrogate benchmark search space is built on top of the architecture search space defined by NASBench-101. In the NAS-101 experiment, the search space (collection of possible neural network architectures) is defined with a head (input layer), a body, and a tail (output layer). The body consists of 3 identical block structures with down-sampling modules. Each block structure is characterized by 3 cells, and each cell is represented by the Directed Acyclic Graph (DAG). The architecture design is depicted in Fig. 5.

The DAG representing a single cell consists of at most 7 nodes, with 5 intermediate nodes, 1 input node, and 1 output node. Each node is labeled as one of the fixed operators: max-pooling 3x3, convolution layer 3x3, and convolutional layer 1x1. A solution of the search space is encoded by an upper-triangular adjacency binary matrix 7x7 from the DAG. In our work, we flatten the binary matrix to get a binary vector representation similar to as in [26], which allows us to easily explore the search space.

## 4.2 Experimental Settings

The system architecture can be seen in Fig. 6. To cover the search space better for the Earth Observation data, we needed to train as many models as possible. To be able to train them effectively, we built a model training pipeline.

Pipeline starts with the user’s connection to Login Node via SSH (Secure Shell). From the Login Node, the user is able to trigger training scripts. There are two main scripts the user should trigger to run proper experiments. One of these scripts is, randomly sampling the architectures from NAS-Bench 101 search space and doing a random walk on those sampled architectures. The other main script we have is the training script. This

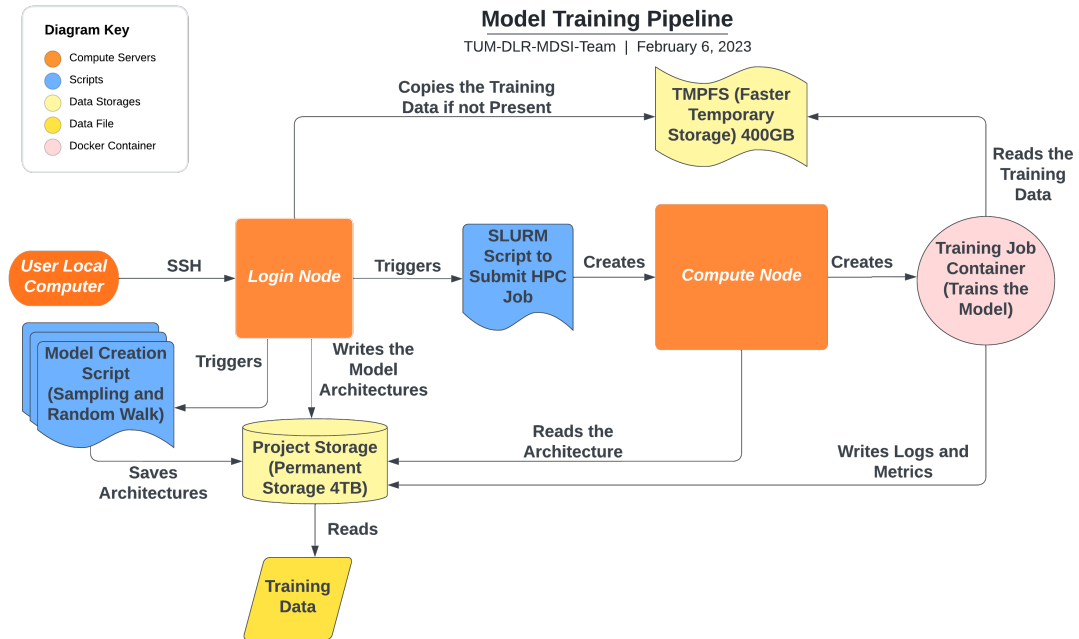


Figure 6: Model training pipeline overview

training script defines the training parameters and runs the model training in DDP (Data Distributed Parallel) manner. The user first triggers the architecture sampling script and samples the architectures randomly. After that, another function in the same script does a random walk with those architectures and saves all the architectures in Pytorch format. As the second step, a batch training script should be triggered. This script triggers the training jobs in batches and submits around 200 individual training jobs to the SLURM. Submitting jobs individually gives us flexibility in fault tolerance in case of a problem in the training of one architecture, and also reduces the time spent in the waiting queue. When the SLURM starts the jobs, it first creates an Apptainer container to run the job. After this container has been created it checks, if the training data has been copied to the faster temporary storage. This is crucial for our process, it makes the training process faster. If the data is not there, the script copies the necessary data to faster temporary storage. Finally, the training job starts and reads the data from faster temporary storage. During training, the job starts writing the training time, training and validation overall accuracies, and training and validation per class accuracies to the permanent storage. In this way, we are able to read the job metrics and compare the results in the experiments.

#### 4.2.1 JUWELS

JUWELS [7] is a supercomputer developed operated by Forschungszentrum Jülich. It is the fastest supercomputer in Europe and the 52nd fastest supercomputer in the world, capable of performing 85 petaflops or the equivalent of 85 quadrillion computing operations per second. It is also the world’s most energy-efficient supercomputer in the highest performance class. JUWELS includes a GPU-equipped compute node called a Booster module. The module features 936 compute nodes equipped with four NVIDIA A100 GPUs with 40GB of memory each, connected via NVLink3 to each other. JUWELS uses a free

and open-source job scheduler for Linux and Unix-like kernels called Slurm. Altogether, we’ve been given 120.000 computing hours to perform the experiments.

### 4.2.2 Training pipeline

The project utilizes PyTorch [15] and PyTorch Lightning [6] to construct the training process, which includes utilizing distributed computing. As the dataset is much larger compared to CIFAR-10, and the original NAS-Bench-101 dataset took a significant amount of time (months) and resources (extensive computing power from Google) to create, the training time for our project will be substantial. To address this, we use multiple GPUs on a single node (machine) to train an architecture and plan to train multiple architectures simultaneously using different SLURM jobs.

**Pytorch DistributedDataParallel.** The DistributedDataParallel (DDP) framework of PyTorch offers the capability for multi-node and multi-GPU training. In the current project, we have implemented the DDP framework to achieve multi-GPU training within a single node for a particular architecture. The `torch.distributed` package provides communication primitives that are essential for parallel processing across nodes, processes, and compute clusters. DDP acts as a wrapper that facilitates synchronous communication among the nodes. The utilization of DDP has two main advantages. Firstly, DDP reduces the time required for data transfer. During distributed training, each spawned process has its own optimizer and performs optimization independently. This eliminates the need to transfer tensors between nodes as each process holds the same gradients. Secondly, DDP removes the overheads of data parallelizing that can arise due to the interpreter’s requirement to copy models, perform multi-threading, and run processes from a single “master” GPU. This is particularly critical for large models with many recurrent layers.

**Utilizing DDP.** Before training, each GPU uses a separate process (multi-threading), and there are 4 GPUs available per node. The dataset is divided into 4 sub-mini-batches with an equal size for each GPU. In the forward step, each GPU independently performs a prediction, and their results are then aggregated. Fig. 7 illustrates the preparation and forward step for distributed training in our project. In the backward pass of the non-distributed data parallelizing version, a “master node” is required to collect all the outputs, calculate the gradient and distribute it to all of the replicated models on each GPU. This centralized mechanism creates an additional overhead, which is eliminated by DDP. In DDP training, each GPU is responsible for communicating its own gradients calculated by the corresponding sub-minibatch to the other GPUs. After all GPUs in the node have received the full set of gradients, each GPU then aggregates the final result, and its respective replicated model will receive the same update. This method is known as **all-reduce** operation. Figure 8 shows the difference between the all-reduce operation (right) and centralized aggregation (left) that causes overhead in synchronizing operation. The DDP back-propagation step in our project is illustrated in Fig. 9.

To achieve a distributed environment, several important parameters need to be defined. Firstly, an IP address and a port need to be defined for the processes to communicate with

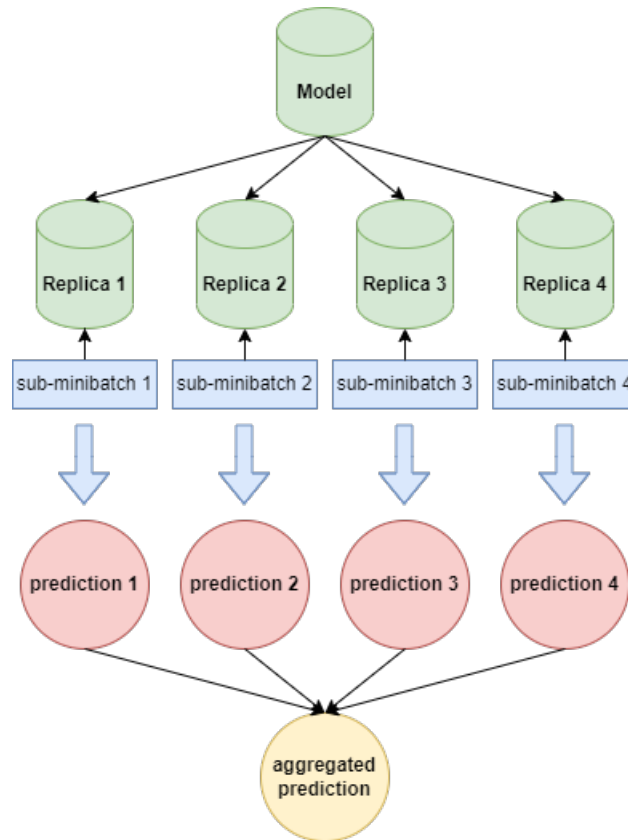


Figure 7: DDP forward pass with 4 GPUs

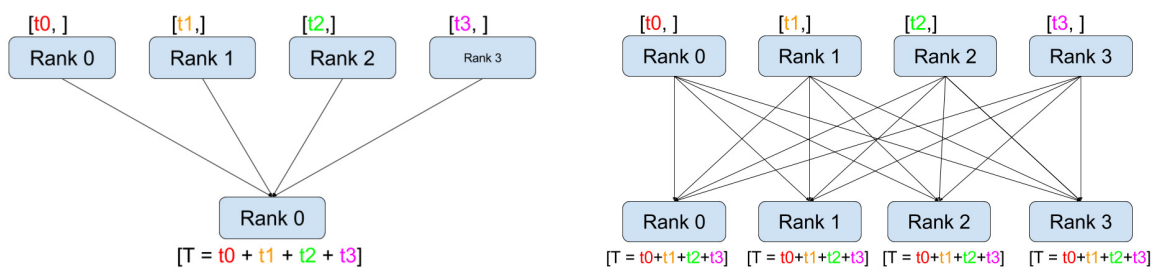


Figure 8: Centralized Reduce (left) versus All-Reduce operations [15]

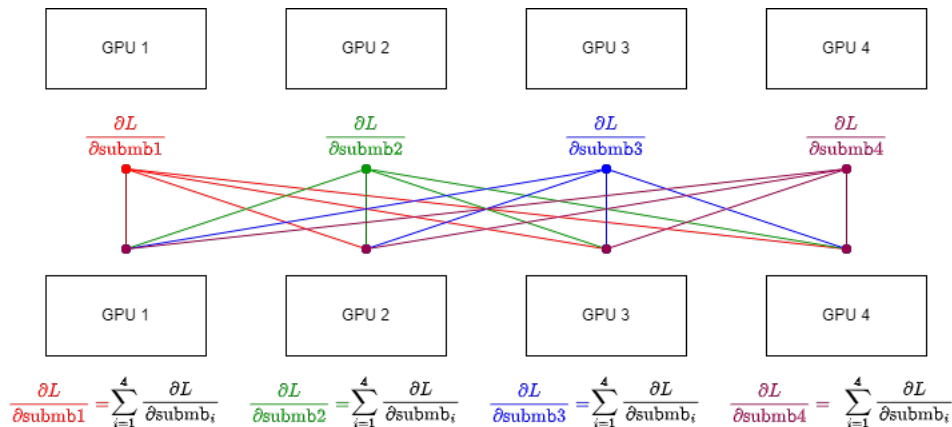


Figure 9: DDP backward pass with 4 GPUs. At the end of the backward step, each GPU receives the sum of all gradients, thus each model has the same final update. *submb* is the abbreviation for sub-minibatch.

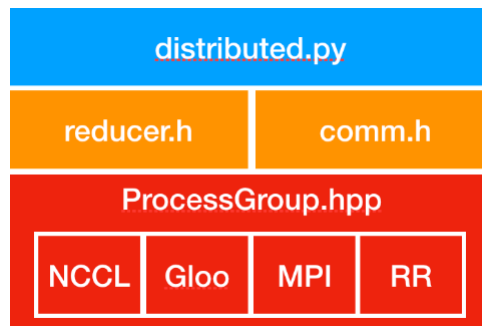


Figure 10: Components of Pytorch DDP [15]

each other. Then, the environment enables the processes to be spawned and exchange information by specifying the following parameters:

1. **Rank:** decides whether a process is a “worker” or a “master” process. There are concepts of local rank (ranks of GPUs within a node) and global (ranks of GPUs across all the nodes). Since we leverage the multi-GPU per node setting, we only care about the local ranks. The process with rank 0 is automatically assigned as the master process.
2. **World size:** corresponds to the total number of processes to be utilized. This variable tells the master about the number of workers to have queued for the synchronization process. It is calculated by multiplying the number of GPUs by the number of nodes. For our project, the world size is always 4 (4 GPUs and 1 node).
3. **Master Address:** IP address of the machine that hosts the process with rank 0.
4. **Backend:** the platform that supports communication between the processes.

**DDP with PyTorch Lightning.** (PyTorch) Lightning [6] is a lightweight framework for PyTorch that allows fast and easy experimentation with deep-learning models. It

provides a high-level API for defining and training models, with built-in support for distributed training and other features. The framework is designed to be highly modular, making it easy to add custom functionality and integrate with other libraries. It also aims to minimize boilerplate code and reduce the amount of time needed to get a model up and running. Overall, Lightning aims to make it simpler and more efficient to develop and train models using PyTorch. Given that there is no member of our team who is proficient in implementing distributed training in PyTorch, it is probable that we may encounter unforeseen bugs when configuring the parameters and establishing the distributed environment. Additionally, our training pipeline is restricted to a basic image classification task utilizing the cross entropy loss. Therefore, we convert our training pipeline to PyTorch Lightning, making use of its proven and dependable implementation of DDP which is recognized by the research community. Lightning’s DDP has been widely used and benchmarked in various real-world projects. It provides a high-level API for distributed training and abstracts away many of the low-level details, making it simpler to use DDP in practice. Additionally, PyTorch Lightning provides built-in support for many advanced features such as early stopping, and callback functions that can further improve the reliability of distributed training. Hence, it requires a minimal effort of one line of code to activate DDP after the pipeline has been developed. Fig. 10 illustrates different types of backend supported by PyTorch in the red box. We use NCCL [15] backend for parallel GPU computing as it provides the optimal communication between CUDA Tensors.

There are several key considerations for DDP training. The first factor is the number of workers (processes) used for simultaneously loading data into RAM for fast processing. While this can increase the efficiency of GPU usage, having too many workers can result in a bottleneck due to additional overhead of communication between the workers. In our experiments, using 8 workers for data loading has been found to provide a significant improvement in terms of training time compared to single GPU training. In addition, when training with multiple GPUs, the batch size needs to be reduced in order to achieve the same results as training with a single GPU. For example, the performance of the following two settings are equivalent:  $batch\_size=1024, gpus=1$  and  $batch\_size=256, gpus=4$ . Finally, the learning rate is linearly scaled with the number of GPUs to achieve the same results of the normal setting with 1 GPU.

### 4.3 Training Setting

For the training, we tried to use the corresponding parameters from the NASBench-101 to allow for a reasonable comparison.

- **Batch size:** 256 as in [31].
- **Number of workers:** 8, which, after heuristic benchmarking per epoch in many settings, 8 workers resulted in the shortest training time for many batch sizes, including 64, 128, 51, 1024 and 2048.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum 0.9 and weight decay  $5e - 4$ .

- **Architecture Sampling Key:** We set this key as 42, which is used during the architecture sampling process.

We evaluate 1000 architectures using validation accuracy, similar to the approach taken in NASBench-101 paper. Additionally, we compute and keep track of the accuracy per class, and average micro accuracy to gain insight into the training process, given the imbalanced nature of the data. Finally, all the models are trained using DDP with 4 GPUs.

## 5 Results

We perform a quantitative landscape analysis of the So2Sat LCZ42 search space in comparison to the CIFAR10 dataset. Altogether we use 907 fully trained architectures on the So2Sat LCZ42 dataset and the corresponding classification accuracies obtained on CIFAR10. Training each architecture took an average of 2.54 hours. To evaluate the landscape, we’ve used an aggregation of the 8 landscape evaluation metrics called Footprint described in Sec. 2.4.1. This is depicted in Fig. 11.

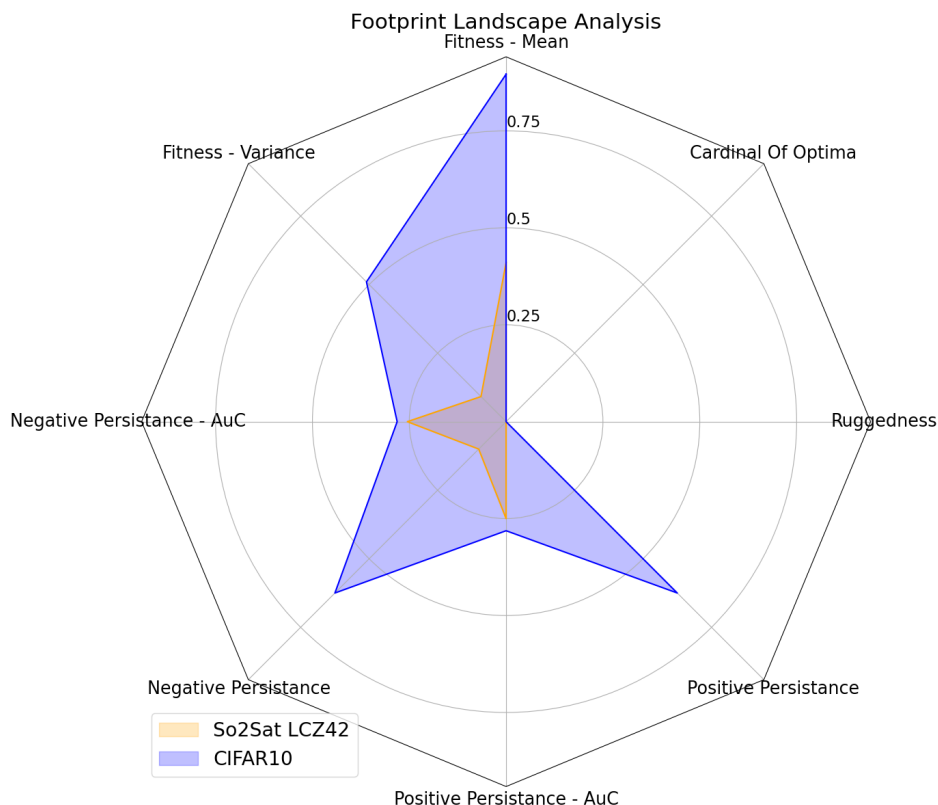


Figure 11: Footprint analysis of the NAS landscape on the CIFAR10 and So2Sat LCZ42 dataset

The results indicate that So2Sat LCZ42 is a more challenging landscape compared to CIFAR-10. The identical models are able to achieve significantly higher validation accuracy on the CIFAR-10 dataset than on the So2Sat LCZ42 dataset. Furthermore, the elite and the worst-performing models persist over time substantially more often than on the

CIFAR-10 dataset. This allows for the early stopping of the model training on CIFAR-10, potentially reducing the computational time to explore the search space. However, higher variance suggests more fluctuations and difficulties needed to be overcome by local search-based algorithms on CIFAR-10 than on the So2Sat LCZ42 dataset.

Besides analyzing the landscape through the Footprint metrics, we’ve also computed macro and micro classification accuracy statistics. Comparison between So2Sat LCZ42 and CIFAR-10 landscape can be seen in Tab. 1. The result of a significant difference between micro and macro classification accuracies on the So2Sat LCZ42 dataset indicates a class imbalance of the dataset. This is in contrast to the perfectly balanced CIFAR-10 dataset. Furthermore, higher micro classification accuracy shows that the trained models tend to focus on the most represented class, unable to deal well with the class imbalance. These results further support the claim that the So2Sat LCZ42 landscape is significantly more challenging.

	Macro Acc.↑	Micro Acc.↑
So2Sat LCZ42	41.34%	58.77%
CIFAR-10	89.62%	89.62%

Table 1: Comparison of the average micro and macro classification accuracy between the So2Sat2 LCZ42 and CIFAR-10 landscapes

## 6 Conclusion

Automated Machine Learning and Neural Architecture Search have made many advances in the fields of natural language processing and computer vision [5], significantly reducing the cost of designing and implementing neural network architectures. However, the study shows [25] that existing NAS benchmarks are difficult to be applied directly to the Earth observation data analysis. To fill the gap in NAS in the field of Earth observation, we have created and open-sourced a complete pipeline for the development of the NAS surrogate benchmark for the So2Sat LCZ42 dataset. It is the first-ever NAS surrogate benchmark for Earth Observation data.

We apply surrogate benchmarking on the NASBench-101 search space, given the time constraints and limited computational resources. The benchmark was developed using PyTorch distributed parallel training framework on the JUWELS supercomputer. Through utilizing the framework we saved tens of thousands of computational hours and therefore significantly reducing the environmental impact of the project. We evaluated the search space by Fitness Landscape Footprint - an aggregation of eight general-purpose metrics for quantitative fitness landscape analysis. The result of the analysis indicates that the landscape is significantly more challenging compared to the state-of-the-art surrogate benchmarks using the CIFAR10 dataset. This is especially the case in terms of the mean fitness of the evaluated models as well as the inability of trained models to persist over time on the So2Sat LCZ42 dataset. For future work, we plan to evaluate the surrogate landscape by using a robust surrogate graph convolutional network (GIN) [30] predictor.



## Acknowledgments

This work was supported by the Helmholtz Association's Initiative and Networking Fund on the HAICORE@FZJ partition. Andrés Camero's work is supported by the Helmholtz Association through the Framework of Helmholtz AI [grant number: ZT-I-PF-5-01] - Local Unit "Munich Unit @Aeronautics, Space and Transport (MASTr)".

## References

- [1] Antonio Bruno, Davide Moroni, and Massimo Martinelli. “Efficient Adaptive Ensembling for Image Classification”. In: *arXiv preprint arXiv:2206.07394* (2022).
- [2] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. “Searching for efficient multi-scale architectures for dense image prediction”. In: *Advances in neural information processing systems* 31 (2018).
- [3] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794.
- [4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021.
- [5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.
- [6] William Falcon. *PyTorch Lightning*. <https://github.com/PyTorchLightning/pytorch-lightning>. 2019.
- [7] Stefan Kesselheim, Andreas Herten, Kai Krajsek, Jan Ebert, Jenia Jitsev, Mehdi Cherti, Michael Langguth, Bing Gong, Scarlet Stadtler, Amirpasha Mozaffari, et al. “Juwels booster—a supercomputer for large-scale ai research”. In: *High Performance Computing: ISC High Performance Digital 2021 International Workshops, Frankfurt am Main, Germany, June 24–July 2, 2021, Revised Selected Papers 36*. Springer. 2021, pp. 453–468.
- [8] Arjun Krishnakumar, Colin White, Arber Zela, Renbo Tu, Mahmoud Safari, and Frank Hutter. *NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies*. 2022.
- [9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “Learning multiple layers of features from tiny images”. In: *Department of Computer Science, University of Toronto* (2009).
- [10] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 82–92.
- [11] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. “Progressive Neural Architecture Search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

- [12] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. “Hierarchical Representations for Efficient Architecture Search”. In: *International Conference on Learning Representations*. 2018.
- [13] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).
- [14] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. “Evaluating Efficient Performance Estimators of Neural Architectures”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. 2021.
- [15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.
- [16] Erik Pitzer and Michael Affenzeller. “A Comprehensive Survey on Fitness Landscape Analysis”. In: *Recent Advances in Intelligent Engineering Systems*. Ed. by János Fodor, Ryszard Klempous, and Carmen Paz Suárez Araujo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 161–191.
- [17] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. “Regularized Evolution for Image Classifier Architecture Search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence 33.01* (July 2019), pp. 4780–4789.
- [18] Robin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. “Speedy Performance Estimation for Neural Architecture Search”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 4079–4092.
- [19] Michael Ruchte, Arber Zela, Julien Siems, Josif Grabocka, and Frank Hutter. *NASLib: A Modular and Flexible Neural Architecture Search Library*. <https://github.com/automl/NASLib>. 2020.
- [20] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. “Taking the human out of the loop: A review of Bayesian optimization”. In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.
- [21] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. “Nas-bench-301 and the case for surrogate benchmarks for neural architecture search”. In: *arXiv preprint arXiv:2008.09777* (2020).
- [22] I. D. Stewart and T. R. Oke. “Local Climate Zones for Urban Temperature Studies”. In: *Bulletin of the American Meteorological Society* 93.12 (Dec. 2012), pp. 1879–1900.

- [23] Kevin Swersky, Jasper Snoek, and Ryan P Adams. “Multi-Task Bayesian Optimization”. In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013.
- [24] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.
- [25] K. R. Traoré, A. Camero, and X. X. Zhu. “LANDSCAPE OF NEURAL ARCHITECTURE SEARCH ACROSS SENSORS: HOW MUCH DO THEY DIFFER ?” In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences V-3-2022* (2022), pp. 217–224.
- [26] Kalifou René Traoré, Andrés Camero, and Xiao Xiang Zhu. “A Data-driven Approach to Neural Architecture Search Initialization”. In: *CoRR* abs/2111.03524 (2021).
- [27] Kalifou René Traoré, Andrés Camero, and Xiao Xiang Zhu. “Fitness Landscape Footprint: A Framework to Compare Neural Architecture Search Problems”. In: *arXiv preprint arXiv:2111.01584* (2021).
- [28] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. *How Powerful are Performance Predictors in Neural Architecture Search?* 2021.
- [29] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. “Exploring Randomly Wired Neural Networks for Image Recognition”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019.
- [30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019.
- [31] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. “NAS-Bench-101: Towards Reproducible Neural Architecture Search”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 7105–7114.
- [32] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. “Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks”. In: *Tenth International Conference on Learning Representations*. OpenReview. net. 2022, pp. 1–36.
- [33] Xiao Xiang Zhu, Jingliang Hu, Chunping Qiu, Yilei Shi, Jian Kang, Lichao Mou, Hossein Bagheri, Matthias Haberle, Yuansheng Hua, Rong Huang, Lloyd Hughes, Hao Li, Yao Sun, Guichen Zhang, Shiyao Han, Michael Schmitt, and Yuanyuan Wang. “So2Sat LCZ42: A Benchmark Data Set for the Classification of Global Local Climate Zones [Software and Data Sets]”. In: *IEEE Geoscience and Remote Sensing Magazine* 8.3 (2020), pp. 76–89.
- [34] Barret Zoph and Quoc Le. “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations*. 2017.

- [35] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.