D3.4 - ML Methods, Models and Documentation

Work Package	WP3, Data Orchestration and Machine Learning
Lead Author (Org)	HES-SO
Contributing Author(s) (Org)	Lionel Blonde, Pablo Strasser, Grigorios Anagnostopoulos, Alexandros Kalousis (HES-SO)
Due Date	30.11.2023
Date	31.08.2023
Version	V0.1

Dissemination Level

X	PU: Public
	PP: Restricted to other program participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

Disclaimer

Versioning and contribution history

Version	Date	Author	Notes
0.1	31.08.2023	Grigorios Anagnostopoulos, Lionel Blonde, Pablo Strasser (HES-SO)	First complete version of D3.4
1.0	27.03.2024	Lionel Blonde, Pablo Strasser, Alexandros Kalousis	Revised version of D3.4 based on review comments

Terminology

Terminology/Acronym	Description
CSA	Coordination and Support Action
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
ASIC	Application-Specific Integrated Circuit
TPU	Tensor Processing Unit
GPU	Graphical Processing Unit
HPC	High Performance Computing
VQ-VAE	Vector-Quantized Variational AutoEncoder
RVQ	Residual Vector Quantization
bpp	Bit per Pixels
CNN	Convolutional Neural Network
FaaS	Function as a Service
SSL	Self Supervised Learning
LC	Learnt compression

Table of Contents

Table of Figures

Executive Summary

In this deliverable we report on the work done so far within the EO4EU project with an emphasis on the development of generic, task agnostic, machine learning models. We set to develop and deploy within the platform task-agnostic models that will lead to significant reduction of annotation effort required in downstream supervised models and task agnostic models that will allow us to compress to a significant extend EO data. To address these two objectives, we relied upon self-supervised models and learned compression models respectively. In this deliverable, we report on the model training and the evaluation of models that address these objectives. In addition, we also report on the EO4EU infrastructure that allows for the deployment of such models and the provision of inference as a service that the platform makes possible.

As already mentioned in developing the self-supervised models, we aimed at significantly reducing the amount of annotation effort that will be required in downstream supervised learning tasks. The provision of annotations is one of the main bottlenecks in terms of time and human effort, and obviously cost, in developing supervised machine learning models. In the deliverable, we present the training of self-supervised models for Sentinel-2 data and rigorously evaluate them with respect to a downstream supervised learning task; the downstream task is only used as a way to quantify the reduction in the annotation effort that one can expect when using self-supervised models and is in no way used in the training of the self-supervised models. The evaluation results show that we can achieve significant reductions to the annotation effort while staying competitive with supervised models that have been trained on large amounts of annotated data (section 2.2)

In developing learned compression models, we wanted to tackle head-on the challenges that arise from the very large data volumes typically present in many EO data sources, challenges that have to do both with storage as well as bandwidth requirements. Our goal was to learn compression models that result in significant reductions of the data volumes while at the same time preserving to a significant extent the information content of the data. We report on the training and evaluation of the learned compression data on Sentinel-2 (section 2.3). Our evaluation shows that the models achieve significant compression rates, while preserving to a remarkable extent the information content, as this is quantified not only by the reconstruction loss, but also with respect to performance in downstream supervised tasks. At the same time, they produce reconstructions that are superior in visual quality when compared to golden standards such as jpeg.

Section 3 reports on the integration of the developed models in the EO4EU platform and the infrastructure that makes possible the *inference as a service* paradigm for EO data. This is one of the major contributions of the EO4EU platform since it allows its users to easily deploy their trained models on the platform and feed them with appropriate data for inference directly from the source using the data preprocessing and fusion capabilities of the platform.

Section 4 discusses the exploitation of the self-supervised and learned compression models and model architectures in different EO data sources. Such exploitation namely requires adjusting the input layers of the respective models, and in the case of self-supervised models potentially the exploration of different types of augmentations. Adjustments such as these will be rather straightforward to operate when it comes to grid-based EO data, one of the most prominent data structures in EO data. The same section briefly reports on two use-case specific models. The training and reporting on such use-case specific models is treated in greater detail in relevant deliverables in WP5. Finally, in Section 5 we discuss the impact of the machine learning models beyond EO4EU and different usage scenarios.

1 Introduction

1.1 Scope of D3.4

Deliverable "D3.4 - ML Methods, Models, and Documentation" stands as a comprehensive report that distils the essence of the accomplishments of the work undertaken under the focused task "T3.3 - ML-based processing" of WP3 of the EO4EU project. This document casts a spotlight on the ML-related work performed under T3.3, weaving together the model development process, the findings, and the significance of the developed ML models within the realm of EO4EU. Embedded within the context of WP3's overarching mission of Data Orchestration & Machine Learning, this deliverable unpacks the methodologies harnessed within T3.3. It delves into the cutting-edge techniques of Self-Supervised Learning (SSL), that utilise the concept of contrastive learning to offer actionable insights utilising the abundance of unannotated EO data. Moreover, the document presents the advancements brought about by Learned Compression (LC) models, which can play a role in redefining data storage and transmission practices, thanks to models that are tailored to the distribution of the data of interest.

Beyond a mere account of progress in the modelling front, the document accentuates the importance of meticulous documentation practices that characterise T3.3's work. It describes a holistic strategy encompassing metadata provision, development of self-explanatory code, and examples of model usage, complemented by this document's detailed elaboration. This multifaceted documentation approach ensures that the unique models and methodologies birthed under T3.3 are truly accessible, reusable and extensible, thereby paving the way for their strategic application across the diverse EO landscape.

In addition to the detailed modelling and documentation work, which is described in Section 2 of this Deliverable, we also delve deeper into the integration, interaction, and impact of these models in the following sections. Section 3 outlines the way the Machine Learning components are integrated in the technical architecture of the EO4EU platform, highlighting the collaboration between the ML Controller, Wrapper, and Inference Server. Section 4 elaborates on the way that the utility and the performance of these models are demonstrated within EO4EU, by evaluating their performance against common, relevant, existing baselines, and by synergising with EO4EU Use Cases. Moreover, Section 5 discusses the broader impact this work can have, elaborating on the ways users can benefit from them beyond EO4EU, and the democratising effect they have on Earth Observation (EO) data access and processing. Lastly, conclusions drawn are discussed in Section 6, which wraps up this Deliverable.

In summary, the scope of this deliverable is to encapsulate T3.3's efforts within EO4EU, showcasing the potential of Machine Learning for optimising EO data processes. The particular focus is on its two model types: Self-Supervised Learning models and models of Learned Compression. Furthermore, it elaborates on documentation, technical integration, model utility, demonstration of model usage and on the broader impact capacity of this work on democratising EO data processing.

1.2 Relation to other deliverables

Deliverable D3.4 - "ML Methods, Models and Documentation", is inherently related to the rest of the WP3 deliverables, as they all deal with the core technical and modelling work of EO4EU.D3.1

"Knowledge Graph Configuration Manual & Deployment" describes work employed to satisfy the semantic annotation models, bringing to light the semantics of data that can be used by T3.3's ML models. Moreover, D3.2 "EO4EU automated systems and services (a)" which describes the services and systems developed, integrated and tested in EO4EU platform, has an overlap with this deliverable, when it comes to the development and the integration of the ML models. Lastly, there is an interplay with the work reported on D3.5 "Customer facing services increased information uptake (a)" since ML-based models are utilized in the AI/ML Marketplace as a software instance, including the ML models as function calls to the inference server, and are to be used in the customer-facing services, such as in the dashboard-data visualization.

Apart from the inherent relation among the WP3 deliverables, D3.4 is also related to deliverables from other WPs of EO4EU as well. Particularly, in D2.4 "Technical, Operational and Interoperability specifications and Architecture" the exact way that the ML-processing module interplays with the rest of the EO4EU platform and its components is described, while relevant sequence diagrams describe the way a user can use the trained models to execute inference tasks. Moreover, D3.4 is related to D4.1 "Infrastructure & Services Definition (a)" as it concerns multiple aspects of integration planning and services specifications, with the ML module being a part of this ensemble. Moreover, as we describe within Section 4 of this deliverable, the leaders of T3.3 have opted to closely assist the activities of WP5, regarding the Use Case (UC) Definition and Technical Alignment (T5.1) and the resulting Deliverables D5.1 and D5.2 "Pilot Implementation methodology and release of evaluation guidelines" (versions a and b). The motivation was to be able to identify the best way that the proposed models developed within T3.3 can be used to assist EO4EU Use Cases.

2 Development of ML Models

2.1 Introduction

In today's data-rich landscape, Earth Observation (EO) data have become abundant, offering a wealth of information about our planet. These data are vast and varied, painting a detailed picture of Earth's surface and atmosphere over different modalities. They enable a significant number of applications to support, among others, decision making and forecasting. Machine learning tools and in particular supervised ML models play an important role in such applications.

In developing supervised machine learning models in general, and for EO data in particular, the practitioner is very often hindered by a significant hurdle: data is available without accompanying labels or annotations. Typical supervised machine learning models require considerable amounts of annotated data for successful training. Data annotation is arguably one of the most labour and cost intensive parts of the process towards the development of a supervised model. Moreover, these models are always trained for a very specific task on a given dataset; they cannot be used to address different tasks that could be defined on the same dataset and data source. As a result, whenever a new supervised task must be addressed on the same dataset one must go through the same labour-intensive process of collecting data annotations and labels and then train a new model with the new annotations.

Thus, one of the objectives of T3.3, and of the project at large, is the development of generic, task agnostic, transferable models, that do not require annotation and label information for their training. Such task-agnostic models will extract data representations that will be directly transferable and

usable for training supervised models in downstream tasks that are defined on the same dataset. Aside from the model reusability and transferability as it is already obvious of particular interest is the significant reduction of the annotation effort for the downstream supervised tasks (as also described in O4 and the associated KPIs), since as already mentioned this is one of the most time-consuming and costly steps in the development of supervised models. These requirements naturally led to the choice of SSL models and contrastive learning in particular. SSL models are generic and task-agnostic, they extract data representations that significantly reduce the annotation requirements for downstream supervised tasks that are to be defined and addressed on the same dataset.

EO data are characterized by their sheer size and volume, analysing and learning models over them can easily require working with hundreds of gigabytes and terabytes of data. Even transferring this amount of data from the source to where the analysis and model development takes place puts significant strains in terms of communication bandwidth, not to mention the storage requirements[1]. T3.3 set to develop learnt compression models (LC) in order to significantly reduce the data volumes while preserving as much as possible the original information content (O4 and KPIs within there). Such models can significantly reduce the accessibility cost of EO data, if deployed on the data source, as well as the storage and bandwidth requirements within the EO4EU platform and ecosystem.

T3.3 started developing task agnostic SSL and LC models early in the project, in order for the first models to be available within the set time frame together with a thorough evaluation of their properties and performance. To be able to design, develop, and evaluate the SSL and LC model architectures a particular data source had to be chosen, since these are tasks that are dataset specific. We chose the Sentinel-2 data source because it is one of the most widely used EO4EU data sources and exhibits structural properties that are shared by many other sources (many channels, spatial structure). An additional motivation for the choice of Sentinel-2 was the availability of open annotated data for well-defined downstream tasks (BigEarthNet land usage). The annotated data and the respective learning tasks allow us to perform a more meaningful evaluation of the SSL models with respect to the reduction in annotation effort that these models bring for potential downstream supervised learning tasks. It should be stressed that the BigEarthNet annotations and labels are only used to provide a downstream task which we use to evaluate the quality of the learnt SSL representations and how they fair with respect to different amounts of annotated data. The BigEarthNet annotations *are not* used and *are not required* in *any way* in the training of the SSL models. The developed SSL models can be used for *any* downstream task that seeks to train supervised models on Sentinel-2, such as regression, classification or segmentation tasks. The evaluation of the LC models does not require the availability of downstream tasks since one can quantify their performance using the error between the original and compressed data. However, we have chosen to start also with Sentinel-2 since this allowed us to share parts of the SSL and LC model architectures.

Even though the SSL and LC models are task-agnostic, they are obviously not data agnostic, they can only be used on data that have the structure and the semantics of the data on which they have been trained. To adapt these models to other data sources one needs to adapt the input layer(s) in the respective model architecture(s) to the data structure of the new data source on which we want to train the model and then train and evaluate the models on the new data source. Note that the training of both model types (SSL, LC) *does not require annotated data* and can be done in a downstream task-

[1] For example training models for Use case 5, soil erosion, required the transfer and preprocessing of 10TB of Sentinel-2 data. Transferring these data over 10Gbit line required several days. In fact to exploit all the available data one would need to transfer 300TBs, which could easily need one month. The LC models that we developed for Sentinel-2 compress the raw data up to one order of magnitude without significant loss of information and visual quality.

agnostic manner. The trained models are made available in the inference server for use by *any* downstream task leveraging features extracted by one of the deployed SSL models or facilitating heavy data transfer (LC) from the respective data source and/or within the EO4EU platform.

The process of model development, and the deployment of these models in the EO4EU platform underscored and guided the development of all the platform components that make inference as a service possible, from data preprocessing and data preparation for inference, to inference itself. This enabled the use of the inference results downstream for different applications. The availability of executable inference ML models on the EO4EU platform together with the data fusion and data processing components instantiates the "inference as a service" framework and is one of the core contributions of EO4EU. The users of the platform, and the EO community, can directly use the inference services within their applications, either through the platform, using the platform to create workflows that prepare and structure data for inference and then feed them to the models running in the inference server.

2.2 Self-Supervised Learning (SSL) models

2.2.1 Introduction to Self-Supervised Learning

In recent years, contrastive learning has rapidly emerged as an essential element of the machine learning practitioner's toolkit, mainly due to annotation-efficiency benefits in low data regimes. In essence, contrastive learning is a machine learning technique that can learn the general features of a collection of data points without any annotations or labels. It works by telling the model about similarities and differences between points in the dataset, while not giving any hint to the model as to what characterizes each point in isolation (e.g. to what category such a point might belong). The model learns features about a data point only relatively to the other points (and augmentations thereof, as we will see below). The features learnt by SSL can then be used in downstream learning tasks, e.g. a supervised learning task, and significantly reduce the amount of supervisory information required compared to the standard learning approaches that do not exploit SSL learnt features.

Similarly to how the human brain can learn high-level, general features simply by trying to recognize what makes two scenes different and similar, an AI model trained on EO by contrastive learning can extract general features if it is trained to tell how different or similar earth scenes are. For example, by "contrasting" EO containing healthy forests with ones containing burned areas, the model has the means to extract and learn what features characterize either type of scenes, while also learning (providing enough data points are fed to the model) their common features (cf. Figure 1).

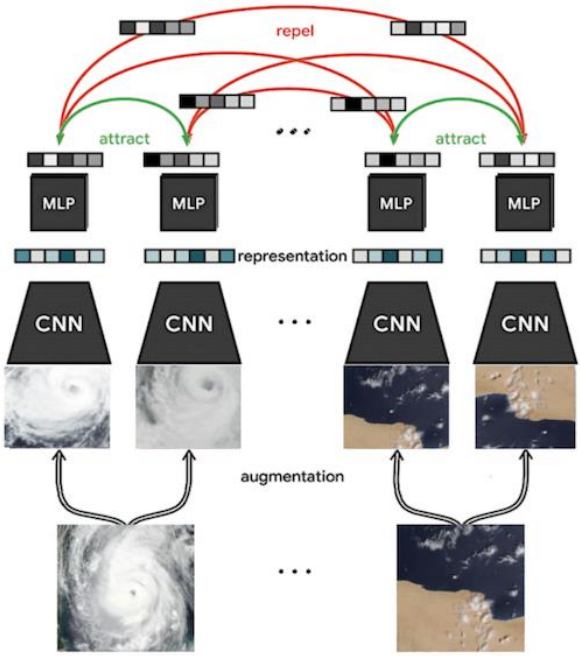


Figure 1: SimCLR contrastive self-supervised architecture with EO inputs[2]

Importantly, such a contrastive learner extracts general features simply by contrasting datum pairs; it does not see any annotations or labels like a model trained in a supervised way (e.g., to solve a classification task) would. Since the contrastive model is trained in a completely unsupervised manner, one could say that it learns "by itself" without help from the practitioner. As such, it is "self"-supervised. Such models deal with the absence of labels by carrying out data augmentations such that images can be compared to their augmented selves. The augmentations (also referred to as transformations) are designed such that the desideratum of having an image and its augmentation have similar representation is a sensible thing to want. In such a scenario, the contrastive loss compares the augmented input to their twins, semantically representing the same underlying objects or entities, but under different "views".

The learning methods leveraging contrastive learning that have emerged as the go-to staples in recent years have been SimCLR from the Google Brain team —the original version is SimCLRv1[3], followed by a subsequent version, SimCLRv2, whose pipeline has been significantly scaled up, described in[4]— and Momentum Contrast (MoCo)[5] from Meta (still called Facebook at the time of release). Following the steps of the Google Brain team, the authors of MoCo also proposed an improved version of their technique which they analogously named MoCo-v2[6]. The latter, among other minor additions, notably improves the data augmentation, and adds a projection head similarly to SimCLR's second iteration. Due to their continual revisions and evolutions, these approaches are still considered as being part of the state-of-the-art in self-supervised learning based on contrastive learning. We now give a brief

[2] https://www.earthdata.nasa.gov/learn/articles/ssl-impact-blog

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. International Conference on Machine Learning (ICML), 2020.

[4] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, Geoffrey Hinton. Big Self-Supervised Models are Strong Semi-Supervised Learners. Advances in Neural Information Processing Systems (NeurIPS), 2020.

[5] Momentum Contrast for Unsupervised Visual Representation Learning. Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[6] Xinlei Chen, Haoqi Fan, Ross Girshick, Kaiming He. Improved Baselines with Momentum Contrastive Learning. Tech Report. https://arxiv.org/abs/2003.04297, 2020.

account of the adoption of such approaches in the domain of remote sensing, before describing the method we chose to carry out our investigation.

Self-supervised learning with EO. Even though the appellation of self-supervised is recent, the use of contrastive learning in remote sensing is neither new nor has seen limited adoption. On the contrary, it has been widely used throughout the field[7],[8],[9],[10],[11],[12] and its origins can be traced back to Cheng et al.[13], as outlined in the recent review paper on self-supervised learning in remote sensing from Wang et al.[14]. In the work of Cheng et al., the authors regularized the features of a ConvNet trained for remote sensing scene classification, where the regularization term constraining the features was based on the contrastive loss discussed earlier in this section. Albeit technically using a contrastive cost as a regularization term to shape the intermediary representation vectors towards more general yet salient features, Cheng et al. is not per se proposing a self-supervised learning technique: it proposes a supervised learning technique assisted by an unsupervised, contrastive constraint. By contrast, Tile2Vec[15] seems to be the first pure self-supervised work and leverages contrastive learning to learn a salient image representation for remote sensing image. As the name hints at, Tile2Vec is inspired by Word2Vec.

Standing as a state-of-the-art contrastive learning technique, the data processing and learning mechanism pipeline of SimCLRv2 can be described in three stages through which every input image is piped:

1. **Transform each input image in the dataset into a pair of augmented images**. Each augmented positive pair is to be understood by the model as depicting similar scenes and objects, since the two items of the augmented positive pair are in effect different versions (alternative views) of the same

[7] Lili Zhang, Wanxuan Lu, Jinming Zhang, Hongqi Wang. A Semisupervised Convolution Neural Network for Partial Unlabeled Remote-Sensing Image Segmentation. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2021.

[8] Juntao Yang, Zhizhong Kang, Ze Yang, Juan Xie, Bin Xue, Jianfeng Yang, Jinyou Tao. A Laboratory Open-Set Martian Rock Classification Method Based on Spectral Signatures. IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium, 2020.

[9] Weiqiang Rao, Ying Qu, Lianru Gao, Xu Sun, Yuanfeng Wu, Bing Zhang. Transferable network with Siamese architecture for anomaly detection in hyperspectral images. International Journal of Applied Earth Observation and Geoinformation, 2022.

[10] He Jing, Yongqiang Cheng, Hao Wu, Hongqiang Wang. Radar Target Detection With Multi-Task Learning in Heterogeneous Environment. Conference on Robots and Vision, 2021.

[11] Lamei Zhang, Siyu Zhang, Bin Zou, Hongwei Dong. Unsupervised Deep Representation Learning and Few-Shot Classification of PolSAR Images. IEEE Transactions on Neural Networks and Learning Systems, 2018.

[12] Wanxuan Geng, Weixun Zhou, Shuanggen Jin. Multi-View Urban Scene Classification with a Complementary-Information Learning Model. Photogrammetric Engineering & Remote Sensing, 2022.

[13] Gong Cheng, Ceyuan Yang, Xiwen Yao, Lei Guo, Junwei Han. When Deep Learning Meets Metric Learning: Remote Sensing Image Scene Classification via Learning Discriminative CNNs. IEEE Transactions on Geoscience and Remote Sensing, 2018.

[14] Yi Wang, Conrad M Albrecht, Nassim Ait Ali Braham, Lichao Mou, Xiao Xiang Zhu. Self-supervised Learning in Remote Sensing: A Review. Accepted by IEEE Geoscience and Remote Sensing Magazine, 2022.

[15] Neal Jean, Sherrie Wang, Anshul Samar, George Azzari, David Lobell, Stefano Ermon. Tile2Vec: Unsupervised Representation Learning for Spatially Distributed Data. AAAI Conference on Artificial Intelligence, 2019.

image. To augment an image, the latter is fed through a stack of image transformations carefully designed by the practitioner to allow the contrastive model to grasp the relevant features for the widest range of potential downstream tasks, while remaining invariant to features that are irrelevant for any of the said potential downstream task. For example, to allow the learner to understand the concept of burned area, allowing the stack of transforms to involve a grayscale filter is probably something to avoid as the model would otherwise be encouraged (by the practitioner) to think that colours are an irrelevant feature for the unsupervised task of contrastive learning. In terms of inputs for the unsupervised contrastive task, the augmented pair for each input image (likely different augmentations since at least one transform in the stack is stochastic in nature) form a positive pair, while negative pairs are obtained by creating pairings between either of the two augmented images and every other image in our minibatch. The mini-batch size (set as a hyper-parameter of the machine learning task) therefore plays a significantly greater role: it determines the ratio of positive and negative pairs that are available to the contrastive model at a given step. Imbalances in positive-negative pairings naturally cause numerous issues in training dynamics.

2. **Feed the pairs to a deep neural network backbone** (each image of a pair goes into the same feature extractor modelled with the said neural network) and obtain a (feature) vector representation of each image as the result of the extraction. The "ResNet" architecture[16] is a prime candidate.
3. **Minimize the contrastive loss**. It involves both positive and negative pairs, and its optimization pushes the contrastive model towards returning similar representations for similar images (positive pairs) and different representation for different images (negative pairs). Through the lens of metric learning, Sohn introduced the "NT-Xent" loss, or Normalized Temperature-scaled Cross Entropy Loss, which is a prime candidate for the loss function used to solve the contrastive learning task[17]. In addition, it is possible to fend off and counteract the imbalances in positive- negative pairings by applying a data-dependent scaling rule at the level of the loss function as it is being optimized.

By being trained in such a way, a contrastive SSL model learns vector representations that enable it to distinguish between images by grouping similar ones closely together in this vector representation space, without ever being told what those images are or contain. This ability is particularly appealing when labels are scarce and data points are bare, without any annotation, which is a situation often met when labelling is expensive or technologically difficult to obtain for the data points.

Due to its ability to capture vector-embedded semantics from unannotated data, self-supervised learning, and specifically contrastive SSL learning, naturally shines in semi-supervised learning: the label-scarcity-facing practitioner can use a self-supervised learning approach as a pre-processing step. First, the model learns general features from (a priori abundant) unlabelled data by attempting to solve a self-supervised contrastive learning task. Then, the practitioner fine-tunes the parameters of the "pre-trained" neural network model by subsequently trying to solve a (supervised) image classification task using the limited labelled data that is available. It has been demonstrated[18] that such a semi-supervised approach is very label efficient. Self- supervised learning enables the practitioner to leverage large amounts of unlabelled data to learn general features that can then be refined for a supervised task for which labels are scarce. It is also not uncommon for this two-stage label-efficient approach to surpass purely supervised methods (cf. Figure 2).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. Tech Report. https://arxiv.org/abs/1512.03385, 2015.

[17] Kihyuk Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. Advances in Neural Information Processing Systems 29 (NeurIPS), 2016.

[18] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. International Conference on Machine Learning (ICML), 2020.

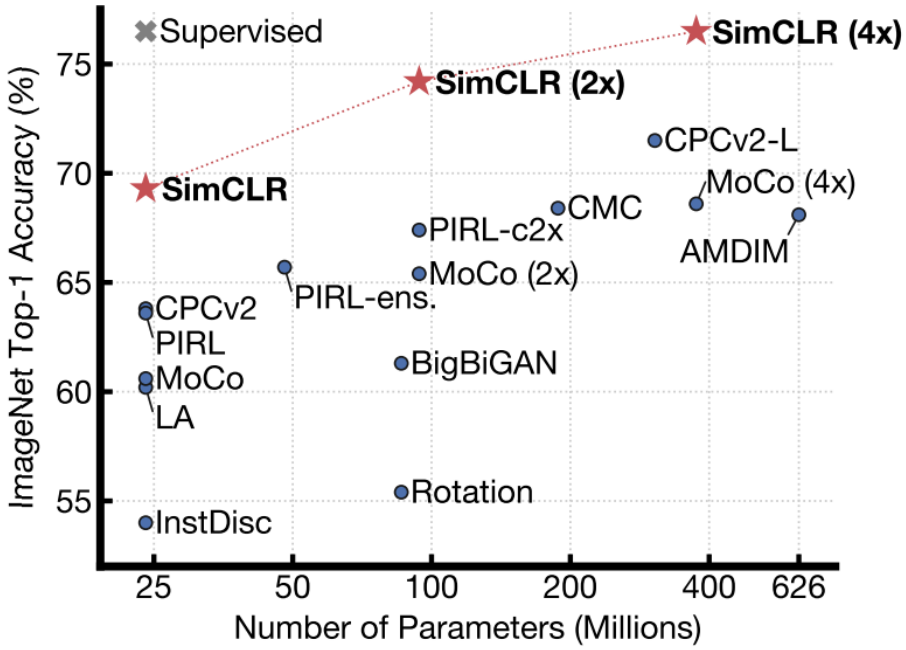


Figure 2: How SimCLR fares among scaling laws of various algorithms in terms the evolution of ImageNet Top-1 Accuracy with respect to the number of parameters[19]

In the context of the EO4EU project, as outlined in the proposal, we have set out to take the self-supervised learning route to benefit from the label-efficient training it can boast in certain domains. Despite being inherently intricate and complex through the lens of computer vision, the satellite images that are dealt with in remote sensing usually display a high degree of repetition of small motifs and low-level visual features that self-supervised models have been shown to excel at isolating in their learned feature bank. The self-supervised learning models that we have deployed through the inference server set up in the context of the EO4EU platform have been implemented following the teachings of the main players in this sub-field of machine learning. Notably, we adopted, and adapted when needed, the self-proclaimed "simple" modelling paradigm of SimCLR described earlier. We made it a point to approach the state-of-the-art practices as closely as possible and found that the only hurdles left to cross were hardware related. Within the boundaries of the project, this frontier will not be crossed (e.g., through the usage of ASIC, or Application-Specific Integrated Circuits, such as Tensor Processing Units, abbreviated TPU). On that front, we might still extend the models in directions not yet explored such as model and pipeline parallelism as we gain access to more sophisticated HPC-enabled infrastructures. In its current form, the models are already implementing a data parallelism paradigm, allowing them to load and flush data samples faster to and from GPU memory, where the heavy-lifting matrix-multiply-based processing is carried out throughout the entire learning process.

2.2.2 Data type selection and data preparation

For the task of learning salient hidden representations via self-supervision, we opted for the use of the multispectral Sentinel-2 satellite images, since they are some of the most popular EO data available. As already discussed previously, in order to meaningfully evaluate the SSL models in terms

[19] Chen, Ting, et al. "A Simple Framework for Contrastive Learning of Visual Representations." *International Conference on Machine Learning (ICML)*, 2020, http://arxiv.org/abs/2002.05709.

of their performance and the reduction of annotation effort they bring, we had to rely on some downstream task. We turned to the BigEarthNet (BEN) dataset to provide early-on in the project the context and the annotations for a meaningful evaluation of the SSL representations. The BEN annotations are not used in any way in the training of the SSL representations. They are used only to provide the downstream task evaluation context. The SSL representations can be used on any downstream task that involves producing learning models on Sentinel-2 data. For example, use case 5, soil erosion, builds predictive models for soil erosion that use as input Sentinel-2 data. Within this use case a first baseline machine learning model has already been developed and deployed on the platform, and we will follow-up with new versions of the models that make use of the SSL features. Since training SSL models does not require annotated data, developing such models for other data sources namely requires adjusting the input components of their architectures to the data structures of the sources on which they are to be trained. An additional modelling consideration for SSL models is the set of meaningful augmentations for the different data sources. Since a considerable part of EO data are of spatiotemporal nature, the set of augmentations that will be applicable to them will be similar, e.g. rotations, translations, flips, noising and blurring. A case in point is use case 6, locust prediction, which builds forecasting models over ERA5 data. As with UC5, a baseline model that does not make use of SSL features has already been deployed in the EO4EU platform. We will also train SSL models for ERA5 data, make them available in the platform, and use them in the newer versions of UC6 models.

The BEN dataset, which we use to benchmark the annotation efficiency of SSL features on Sentinel-2, is a well-known reference benchmark in remote sensing. It is a curated annotated collection of multispectral Sentinel-2 satellite images. The BEN dataset is a land-usage classification dataset that consists of 590326 Sentinel-2 image patches annotated by multi-labels for RS image understanding, but the authors note that, by visual inspection, they have identified that 70987 images are fully covered by seasonal snow, cloud and cloud shadow. Following the same tracks as the authors did when they used the BEN dataset, we filter out these images before using the dataset for machine learning tasks. We consider that these images are corrupted, but they might prove useful for other types of tasks. For benchmarks to be somewhat comparable, we therefore use the exact same splits (train, validation, and test) as they did in their codebase. The standard version of the BEN dataset presents samples with 43 labels. Still, the authors give the [20] to use 19 clusters of these labels instead, effectively simplifying the task at hand from a 43-label classification to a 19-label one. We considered the standard 43-label setting first and extended the code to handle the 19-label setting as well later. In our experiments laid out below, we report on the 19-label setting because we deem the class imbalance present in the 43-label setting too severe for the results to be easily interpretable.
The BEN dataset comes with a set of data-loading and preprocessing routines[21]. We took inspiration from their codebase when it comes to how to pre-process the data so that they can be readily fed in a machine learning pipeline. We adapted these procedures and made them available as data preprocessing steps in the EO4EU pipeline (fusion component) to prepare Sentinel-2 data for training as well as for inference for SSL.

Multi-class vs. multi-label: in multiclass classification, each data point belongs to one and only one of the many classes or categories that might relevant. The task involves assigning a single label to each data point from a set of multiple possible labels. The goal is to correctly predict the most appropriate

[20] Sumbul, Gencer, et al. "BigEarthNet-MM: A Large Scale Multi-Modal Multi-Label Benchmark Archive for Remote Sensing Image Classification and Retrieval." *arXiv [cs.CV]*, 17 May 2021, http://arxiv.org/abs/2105.07921. arXiv.
[21] Sumbul, Gencer, et al. "BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding." *arXiv [cs.CV]*, 16 Feb. 2019, https://bigearth.net/static/documents/BigEarthNet_IGARSS_2019.pdf. arXiv.

class for each input. By contrast, in multilabel classification, each data point can be associated with multiple labels simultaneously, which are not mutually exclusive as in the multi-class case. The task involves predicting a set of labels for each data point, where the labels can be from a predefined set of categories. This is particularly useful when an item can belong to more than one category or class. Since BEN is originally designed to benchmark multi-label classifiers, we are dealing with samples to which the associated label vectors can be represented as multi-hot vectors, since each sample often belongs to more than one class simultaneously. This is in contrast with the more common one-hot vector representation used when dealing with the considerably easier multi-class setting (subsuming multi-label). In practise, this means that instead of modelling the output of the model (deep neural network) with softmax function of dimension 43 —which transforms a vector of logits of size 43 into a normalized vector that can be interpreted as a probability vector over the 43 labels— we model pass the 43 output logits through an element-wise sigmoid vector. In effect, this does not transform the vector of logits into a probability vector over the labels like is desired in the multi-class setting, but instead transforms each of the 43 logits into what is interpretable as a probability (since the sigmoid squashes its input between 0 and 1). This is more suitable for the multi-label setting we tackle.

Heterogeneous input image resolutions: as in Sentinel-2 convention, in the BEN dataset as well, an input is a collection of 13 bands. Concretely, in the BEN dataset each of these bands is a TIFF file (technically, a GeoTIFF file —containing geo-location overlay with the pixel content of the image). Each collection of 13 bands is associated with a JSON file which contains metadata and the label (technically, one or more labels) associated with the said input. The training set is far too big to fit in memory entirely; we therefore designed the data loader around this constraint and constructed it such that it would iterate (in a distributed fashion) over thousands of files on the fly as the model learns in minibatch. This means that only a small portion of the training data is treated by the learning algorithm at a given iteration throughout the learning process. The 13 bands correspond to geo-aligned bands from the Sentinel-2 data source (cf. Figure 3: Sentinel-2 bands). While aligned in terms of the geographic area that they cover for a given instance, they are not the same size in terms of pixels; said differently, not all the bands are depictions of a given area with the same resolution. The discrepancy in resolution between bands (a total of three different resolutions among the 13) can be addressed in various ways when tasked with feeding the data into a machine learning pipeline.

The original attempt to solve the problem of classification with BEN (whose authors are the researchers who released the dataset) involved the design of a model with several (learnable) features extractors organized in branches, one per resolution, for a total of three branches in total, before being aggregated into a unified trunk, and processed further by a shallow neural architecture. This branch-based architecture can be found online in one of the codebases released by the originating lab, along with their most recent architecture which abandons the idea of processing the bands per resolution and working with branches. Instead ---and this is the approach we adopt--- the three channels of resolution 60[22] are eliminated, and the bands of resolution 20 are upscaled (by bilinear interpolation with OpenCV) to match the resolution 10. To learn more about each of the bands, refer to the following figure describing the spectral bands of the Sentinel-2 sensors, extracted from the "Sentinel-2" Wikipedia page[23]. In effect, 10 bands remain, each with the same size/resolution. These images can therefore be aggregated along the batch dimension (i.e., stacked on top of each other structurally, without merging their respective contents), such that the initial 13-band input collection with size/resolution mismatch now boils down to a 10-dimensional stack of resolution-matching bands. This stack is concretely treated as a tensor by our PyTorch-based codebase, where each of the

[22] Sumbul, Gencer, et al. "BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding." *arXiv [cs.CV]*, 16 Feb. 2019, https://bigearth.net/static/documents/BigEarthNet_IGARSS_2019.pdf. arXiv.

[23] https://en.wikipedia.org/wiki/Sentinel-2

10 input bands is treated as an input channel in a 10-by-120-by-120 input tensor. These tensors are the inputs of our models.

Sentinel-2 bands	Sentinel-2A		Sentinel-2B		
	Central wavelength (nm)	Bandwidth (nm)	Central wavelength (nm)	Bandwidth (nm)	Spatial resolution (m)
Band 1 – Coastal aerosol	442.7	21	442.2	21	60
Band 2 – Blue	492.4	66	492.1	66	10
Band 3 – Green	559.8	36	559.0	36	10
Band 4 – Red	664.6	31	664.9	31	10
Band 5 – Vegetation red edge	704.1	15	703.8	16	20
Band 6 – Vegetation red edge	740.5	15	739.1	15	20
Band 7 – Vegetation red edge	782.8	20	779.7	20	20
Band 8 – NIR	832.8	106	832.9	106	10
Band 8A – Narrow NIR	864.7	21	864.0	22	20
Band 9 – Water vapour	945.1	20	943.2	21	60
Band 10 – SWIR – Cirrus	1373.5	31	1376.9	30	60
Band 11 – SWIR	1613.7	91	1610.4	94	20
Band 12 – SWIR	2202.4	175	2185.7	185	20

Figure 3: Sentinel-2 bands[24]

Imbalance in the BEN dataset: the dataset suffers from very severe class imbalance, which renders the training process very noisy and prone to misinterpretations if the evaluation metrics are not computed and amortized over long periods properly. That is the primary reason why we have spent much effort on devising sensible evaluation metrics and techniques. This is also why the in experimental results we report here we have considered the 19-class version of the multilabel task as downstream task; the number of labels were reduced from 43 to 19 to reduce imbalance[25]. We consistently witnessed harsh imbalances as we compute the class imbalances. To allow us to use these computed imbalances, we propagate them to the agent so that they may be used as weights in the evaluation metrics. We added them to the signature of the metrics and integrated the weights into them when we felt it made sense. In essence, we use these "balances" to reweigh the contribution of each class/label in the computed metric, e.g., precision and recall. Intuitively, the model should be penalized less for misclassifying for a label, and it has essentially almost never been activated, compared to a label that is turned on for most of the multi-label instances. This has proven limited success so far because these are numerically hard to tune not to completely hijack the original semantic meaning of the given metric. Still, we left the option open because it showed promise.

Having successfully completed the SSL model development and evaluation on them, we will deploy the selected architectures and models on other EO data such as volumetric data of similar spatial structure, 2D or 3D over multiple channels/bands. As already mentioned, the adaptation of the SSL model architectures to other data sources namely consists in adjusting the input layer(s) to their respective data structures.

2.2.3 Model Evaluation protocol

Downstream task to evaluate the representation learned via self-supervision: To have a meaningful evaluation of the trained SSL model it is imperative to select a downstream, supervised task. The

[24] https://en.wikipedia.org/wiki/Sentinel-2

[25] Sumbul, Gencer, et al. "BigEarthNet-MM: A Large Scale Multi-Modal Multi-Label Benchmark Archive for Remote Sensing Image Classification and Retrieval." *arXiv [cs.CV]*, 17 May 2021, http://arxiv.org/abs/2105.07921. arXiv.

obvious candidate to play the role of such a downstream task to be learned with a model taking the learned representation as input is, multi-label classification based on the BEN dataset. This is exactly how we evaluate the quality of the representation learned by SSL, that was trained without labels, but by simply by contrastive learning with transforms.

Evaluation metrics for self-supervised learning task: The obvious ready-to-use candidate metric to evaluate the performance of a model on the validation set while training is to simply evaluate the loss used to train the model on the minibatch sampled from the validation set. Somewhat reminiscing Goodhart's law (which states that "[w]hen a measure becomes a target, it ceases to be a good measure"), it is a recurring observation in various machine learning subdomains that the training loss is an impaired metric to gauge the quality of the obtained predictions. Specifically, we use transformed inputs in our training loss, as a result of the data augmentation process involved in our contrastive SSL approach. We would like to not involve such transformations in the validation loop, because *"what is the best data augmentation strategy"* is still an open research question. Besides, as the BEN benchmark gives us access to labels, we can leverage those for evaluation purposes without involving them for any training update. To that end, we devised a novel technique to craft an evaluation metric whose purpose is to evaluate the contrastive self-supervised algorithm (as opposed to the downstream task, to which the technique is agnostic). Still, the technique is not computationally cheap (requires a lot of GPU memory) and relies on the construction of a bank of learned representation vectors, which can be costly in terms of time. Consequently, we have been careful not to run such an evaluation sub-loop too often as the time it takes is subtracted from the time the learner has to carry out a training iteration, given that the computational budget allocated to run on the HPC is assigned a fixed and static time limit. The designed evaluation technique articulates as follows. First, the learning agent gathers a bank of features from the current iterate of the model over the dataset. Concretely, as we iterate over the entire validation dataset, we pass input through the model trained via contrastive self-supervision and store the outputs in a collection that we call the feature bank. Among the feature vectors present in the collected feature bank, we then identify the K feature vectors that are closest to each of the feature vectors associated with the inputs in the current evaluation batch. The closeness is computed using a typical similarity metric; in our case, cosine similarity. Since we decided to consider the BEN dataset and use multilabel classification as downstream task, we (not the model) have access to a multi-hot label vector for each of the inputs in the validation dataset. We therefore craft a weighted average of the label vectors for the K closest features for each input in the validation batch, where the weights are the similarity values computed in the previous step. Finally, we use the distance between the label of the evaluation input's feature vector and the similarity-weighted K-average label vector as evaluation metric. The method does use labels, so it cannot be used in scenarios where the practitioner does not have access to a label for each input. In our case, although the learning agent is unsupervised (the general class of methods that contains self-supervised learning), we have access to labels and can use them if we wish as long as we do not use them to influence the unsupervised training process in any way. This evaluation technique, despite requiring access to labels, does not rely on *training* a downstream task that would involve additional training in an inner loop every time we want to evaluate the unsupervised model. Instead, we can easily evaluate the model every fixed number of training iterations, although we limit this number for the sake of time efficiency as explained above. A very valuable trait of this evaluation strategy is that it does not depend on the choice of augmentation strategy, which reduces the risk of confounding factors and interactions and interdependencies between hyper-parameter choices (augmentation stack is one of those).

Evaluation metrics for downstream (multi-label) classification task: our models, materialized as neural networks that remain in GPU memory throughout the entire training process, are difficult to scale up without requiring intricate and highly complex pipeline or tensor parallelism, which are generally hard to set up. In order to use larger networks that would still fit in memory without causing overflows, we used half-precision arithmetic essentially everywhere possible —technically, we make our CUDA tensors mixed-precision, which required the use of an entire machinery consisting in scaling and unscaling the gradient vectors instrumental in the training process of neural network models. This effort to never leave the GPU memory in half- or mixed-precision is extended to the evaluation sub-loop that we orchestrate at regular iteration intervals within the training loop. Concretely, we wanted to perform the inference of evaluation metrics in GPU memory, and in half- or mixed precision. This was trivially done for the devised metric that we describe above, which is a metric we designed specifically for self-supervised learning. When it comes to the evaluation of the downstream task of interest, which as we wrote earlier is the multi-label classification task at the origin of the BEN dataset, the libraries usually used to compute those metrics have not been ported to work in GPU memory. If we were to use them to compute the metrics, we want to evaluate our multi-label classifier, we would first need to transfer the prediction and target tensors (for each batch of data) to the CPU's memory, i.e., the RAM. While less exposed to out-of-memory exceptions, this comes at a cost in speed of computation (and is less elegant). As such, we implemented a GPU-native version of the desired multi-label classification metrics. These metrics are accuracy, precision, recall, F1-score, F2-score, specificity, balanced accuracy (arithmetic mean of recall and specificity —which are the true positive rate and true negative rate), hamming score, and zero-one score. These were implemented specifically for the multi-label case, leaving the option to return the per-label metrics when it makes sense to express it. We implemented a wide array salient metrics to ease further comparison down the line.

2.2.4 Model Training

Choice of data augmentation strategies: we must design transformation stacks to perform the data augmentation transforms at the entrance of the self-supervised learning pipeline. Data augmentation is however only instrumental in the training regime; no transformation is applied to samples from the validation or testing splits during their associated regimes. For a given training instance, the instance is coupled with a transformation of itself by passing the said instance through the training transformation stack. Each transformation is stochastic by design, i.e., it has a random component: passing the same image through the stack twice has a very high probability of yielding a different transformation. The randomness of these transformations is often more complex than simply deciding whether to apply a certain transform or not upon the outcome of a coin flip. Since we are in the multi-label scenario (the downstream task used to evaluate the learned representation is the ancestral multi-label classification task the BEN dataset was introduced for), we do not make use of the transform that is usually part of the data augmentation stack and that consists in taking a random crop of the image. This is because the Sentinel-2 data forming the BEN dataset does not consist of images with centred subjects, with the subject being directly determining the single label to be associated with the image. In our case, it is possible that most of the elements determining several of the label predictions could only constitute a small piece of the input image, and cropping would consistently ensure some crucial items for some of those labels would be cropped away. Cropping would therefore render the task virtually impossible to solve with decent enough precision and recall. Colour jitter and Gaussian blur are the two main transformations that we use on top of the arguably most crucial ones for satellite imagery in remote sensing: random rotations, and random flips (vertically and horizontally), carried out upon the outcome of a coin flip. Figuring out what are the best transformation for remote sensing is still an open question and is subject to be further investigated in the future as we learn more about the impact of each. We are especially curious about characterizing the ones that are the most instrumental in obtaining high performance in various downstream tasks (e.g. multilabel classification for BEN). We now lay out how the discussed data augmentation strategies would fair when addressing the various use cases with self-supervised learning, in view of

reducing their labelling needs. For Use UC5, which leverages Sentinel-2 data as only source of input, the data augmentations we have employed thus far are directly applicable. When it comes to UC6, which uses ERA5, we expect the augmentations used so far to be equally viable. Indeed, despite representing different phenomena and values, the way the data is organised and structured make ERA5, within the context of the ML task to solve, similar to Sentinel-2 with respect to the family of input transformations we want the predictor or the ML task to remain invariant to. We can also consider scenarios beyond the current scope of the use cases, for example tasks operating over sensor data that generate point clouds (e.g. LiDAR). In such a setting, we expect the augmentations like rotations, crops, application of jitter and blur to also be valuable to the saliency of the contrastive learning representation. That is of course dependent on the condition that we would want the said representation to remain invariant to the chosen data transformation to solve the end-task of the use case.

Scaling up the model to large (wide and deep) architectures: once the initial stages of prototyping were over, we quickly scaled up our original models to network sizes that go beyond a hundred layers. Specifically, we model the neural network predictor of SimCLR (the SSL algorithm we use) as a ResNet architecture (He et al., 2015) composed of 101 layers, and commonly referred to as "ResNet101". We chose the size to fully leverage the memory size of the GPU as our disposal, such that a ResNet of 101 would fill approximately 80 to 90% of the GPU video memory. The neural architecture ingests tensors whose construction has been discussed above (10-channel tensors) and return a representation whose size we fix at 64 (typical output size, 128 yield similar results). On top of the ResNet101 convolution-based feature extractor sits a very shallow fully- connected multi-layer perceptron (MLP) that returns the said output representation. This MLP has 2 layers: the first maps the flattened output representation of the convolutional stack of the ResNet101 to a fully connected layer of size 128, and the second (and last) layer maps this 128 penultimate representations to the output one, which is of size 64, as we wrote earlier. PyTorch offers the option to use weights pretrained on the ImageNet dataset as starting point instead of starting off from randomly initialized weights, to which we opt in. Going beyond the network structure itself, the architectures employed involve an entire range of deep learning techniques and tricks that have proven their worth in making model training more stable as well as faster to train. Among the add-ons that stood the test of time are batch normalization, residual or skip connections, weight decay, learning rate schedules, layer-wise learning rate adaption, gradient accumulation, etc. Weight decay regularization notably necessitated particular tweaks. Indeed, special care had to be taken for weight decay as by default it also affects the running mean and standard deviations continually computed and updated for each and every batch normalization layer. While some works in the literature point at the fact that decaying these statistics can lead to increase in performance, we opted for the safe, rational option that consists in turning off weight decay for the parameters that are not multiplicative weights. That includes the running mean and standard deviations of batch normalization layers, as well as the bias vectors across the architecture. Although challenged and discussed in a couple of recent works[26],[27],[28], this is a well-accepted design choice, and it is the design that best aligns with the theory.

Week-long run orchestration: experiments conducted to this point all show that an agent trained with the SimCLR algorithm on the BEN dataset benefits from long training times, until the point where the

[26] Yuxin Wu and Kaiming He. Group normalization. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 3–19, 2018.

[27] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 558–567, 2019.

[28] Summers, Cecilia, and Michael J. Dinneen. Four Things Everyone Should Know to Improve Batch Normalization. International Conference on Learning Representations (ICLR), 2020.

training loss starts stagnating. For practical reasons, we engineered intricate yet robust ways to resume training, so that the same model can continue training over the course of subsequent, yet different jobs orchestrated on a cluster where any individual job can only be scheduled to run for a limited time.

2.2.5 Model finetuning and evaluation

Large batch size accommodation to favour learning with contrastive loss: as highlighted earlier, the use of the NT-Xent contrastive loss in SimCLR makes the learning dynamics particularly dependent on the choice of batch size as a hyper-parameter. Higher batch sizes give the learning more negative examples to contrast the positive ones with at every training cycle. The hurdle we soon arrive at when facing such a direct way to increase performance is how the batch size is limited by the GPU memory at the learner's disposal. To address this issue, we implemented a gradient accumulation subroutine within the training loop, which allowed us to use larger batch sizes in effect, while warding off GPU memory overflow exceptions. The technique consists in summing the gradient contributions of several consecutive training iterations without stepping along the resulting summed vector just yet. We only move the model's weights along the accumulated gradient at regular intervals (as long as it is not as often as making the update every single iteration). By scaling the accumulated loss (and therefore gradient) appropriately, the update is virtually equivalent to carrying out an update with a larger batch size. This structural advancement of the training loop that we designed is responsible for providing much desired benefits, namely an increase in stability. Since at this stage we are using large batch size, we can leverage layerwise learning rate optimizers such as LARS. These are adapting the gradient differently for each layer of the neural architecture, depending on internal statistics and hyper-parameters. In our experiments we have found such an adaptation scheme to be stable and to have a positive impact on the overall performance.

Learning rate scheduling: across our experiments, we made very careful use of cosine learning rate annealing (but without warm restarts) scheduling. It is the scheduling method that was suggested originally for SimCLR, in both the paper and the official implementation on Github[29]. We took particular care of not mishandling such scheduling techniques because they can enter in conflict with the severe class imbalance present in the BEN dataset. A sudden restart of the learning rate schedule (either with restart or inadvertently when continuing a run with another cluster job) on a rare label combination occurrence could have severe consequences. The schedulers also need us to decide beforehand what the total number of epochs is expected to be, which in our case is tough to estimate before witnessing how well/badly the model performs.

Dealing with standard library bugs related to distributed system paradigms: we encountered a technical difficulty in distributing workloads (leveraging a data-parallelism paradigm specifically) with SimCLR due to the size of a batch needed for input data. Self-supervised learning models benefit from large batch sizes and are coupled with their data augmentation, making each batch twice as voluminous in size, as already mentioned above. The classifiers, for example, do not suffer from any PyTorch bug, but SimCLR did and still does because there exists a PyTorch bug (issue on the official Github[30]) that occurs when samples are processed with Python code before being serialized into tensors —which is exactly what must happen in our pre-processing routine explained above due to the necessary assembly of the 10-channel input tensor from the 13 mismatching GeoTIFF band files. This is an issue that causes a slight memory leak but that does compound to GPU memory overflows for large batches such as the ones used in SimCLR. This issue is especially difficult to debug and resolve on distributed HPC clusters because the error messages from Python are erroneously wrapped by

[29] https://github.com/google-research/simclr
[30] https://github.com/pytorch/pytorch/issues/13246#issuecomment-905703662

Pytorch (Python's "multiprocessing" by PyTorch's "distributed"), with the ensuing hardware errors wrapped by Slurm, which is the standard software stack used to orchestrate large HPC clusters (as the one we use). Under such memory constraints, we have so far kept the failure rate low by limiting the number of parallel workers and by asking the cluster to be allocated with GPU with high memory ranges.

What we set out to investigate and want to evaluate in the context of the self-supervised learning impact on EO data is: how very deep classifiers (100+ layers) trained from EO input directly compare, in classification performance, to very shallow (1 layer) classifiers trained from pretrained representations of the same EO data. Importantly, the said representations are the features extracted from the model we have learned by self-supervision, a modelling paradigm completely isolated from the downstream classification problem we want to evaluate. This head-to-head comparison will be declined along an axis describing the number of labels that will be used by the classifier. Namely, we study log-10 granularities: 10%, 1% and 0.1% of labels being available to the classifier. The tasks are all trained and evaluated on the BEN dataset discussed at length above.

Experimental results on how leveraging SSL compensated for the lack of labels: we showcase this result via the comparison between the training of a full deep classifier and the simple training of a single-layer classification head on top of a frozen SimCLR trunk. In the latter, only the 1-layer deep head (called the "probe" or "linear probe") is updated by gradient descent, while the former is a standard classifier. This standard deep classifier has the same number of parameters as the SimCLR trunk combined with the probe (for having consistent comparisons), but the classifier can use the gradients to update the weights of all its layers. Compared to the SSL-based method which only update one layer, the probe, the classifier updates its entire 101-layer deep architecture. This represents a significant difference in compute, which already gives the SSL-based method that we present a meaningful practical advantage. Indeed, relying on less compute means it can be trained faster, a valuable asset for any practitioner. In the experiments presented here, we run the experiments for 10 epochs (one epoch is spent when the learning agent has gone through the entire training set available to it once). We conduct experiments in three regimes of labelled data availability, with the Sentinel-2 EO inputs from BEN: 0.1%, 1%, and 10% of labelled data being provided for training (cf. Figure 4). Another view of the results is depicted in Figure 5, which puts more emphasis on the impact of percentage of label availability on comparative performance. In terms of number of labelled samples, these percentages correspond to supervised learning training sets of sizes 270, 2700, and 27000 images respectively. There is a direct analogy to this setting and the business scenario that set to address in EO4EU, in which we planned to learn "*robust representation[s] that will facilitate downstream tasks minimizing their labelled data budget requirement*." The motivation was described: "*The availability of such representations will relieve to a significant extent the users of the platform that want to develop their specific services from the costly annotation process.*" In the particular case of BEN-like scenario, we could bring this line of thought towards more specific questions: *instead of requiring the user to label 270000 data points, what could be the achievable performance if they only annotated 270 or 27000 data points? Under these constrained budgets of available volumes of labelled data, what can the conventional approach of a deep classifier achieve, and how can SSL help*?

In the figures laid out below, every subplot corresponds to a different classification metric. We remind the reader that the reason why classification metrics are employed here has nothing to do with the self-supervised aspect of the work per se, and everything to do with the downstream task defined to evaluate the representations learned via self-supervision. In our case, tackling Sentinel-2 data, the BEN benchmark allows us to define a classification task as the downstream task that we use to evaluate the results. The metrics outlined on each figure are, ordered from left to right, top to bottom: recall, subset accuracy, balanced accuracy, precision, accuracy, F2-score, and finally F1-score. On every subplot/histogram, the top bar is the linear probe taking the SSL representation at input, while

the bottom row is the deep classifier learned from scratch the inputs. The subset accuracy is specific to the setting of multilabel classification, which is the scenario we tackle here as a downstream task to evaluate the meaningfulness of the learned self-supervised representation. In contrast with the accuracy, which considers the downstream task's model's predictions' accuracy for each label distinctly, as if they were separate binary classification tasks, the subset accuracy returns a success if and only if all the labels of the multilabel setting were correctly predicted for a given instance. As such, if the model is correct in its predictions for all but one label, the prediction will be counted as correct for all but one labels before being aggregated across labels. If the number of labels is high, such an accuracy would then be very close to the accuracy of a model that prediction all labels correctly. This is however not the case for the subset accuracy, which would count a prediction as incorrect if the predictions were not correct for all labels. In scenarios where some labels are particularly difficult to predict accurately, e.g., when the labelled data is not well balanced, the subset is therefore particularly unforgiving compared to the accuracy that considers each label as a separate prediction problem and aggregates the per-label correctness metrics afterwards. Metrics that consider the content of the confusion matrix associated with the downstream task's prediction are a particularly good indicator of those predictions' quality in the considered multifaceted multilabel prediction downstream task. These are metrics whose analytical form involve rates such as true positive, true negative, false positive, and false negative rates. For all the metrics of that type that we consider, the metrics are calculated per label, and aggregated across labels. Precision and recall (also called sensitivity or true positive rate) are a good indicator of the performance for the considered task. The F1-score is the harmonic mean of the precision and recall, and as such give an aggregated metric of the two indicators. The reported F1 scores therefore give a very convincing account of the appeal of our self-supervision-based approach when the labels are scarce. The F2 score is a weighted variant of the F1 score: while the F1 score gives equal weight to recall and precision in the harmonic mean (a weight of 1 for each), the F2 score gives more weight to the recall (a weight of 2 is given to the precision in the denominator of the score's formula). A less common metric we also report is the balanced accuracy, which is the arithmetic mean of the recall (sensitivity, true positive rate) and the specificity (true negative rate). The balanced accuracy is used when dealing with imbalanced data, as it is better equipped to give higher scores to classifiers (again, here only the downstream task best suited for the BEN dataset) that are better at solving the downstream than classifiers that predict 0 for all labels or 1 for all labels. Note that the accuracy (as opposed to the balanced one) is more likely to give higher scores to only-zero and only-one classifiers in imbalanced dataset settings, which is why we consider the balanced accuracy to be a valuable metric in the scenario considered here. Overall, the metrics that synthesize the superiority of the method best are the F-scores and the balanced accuracy.

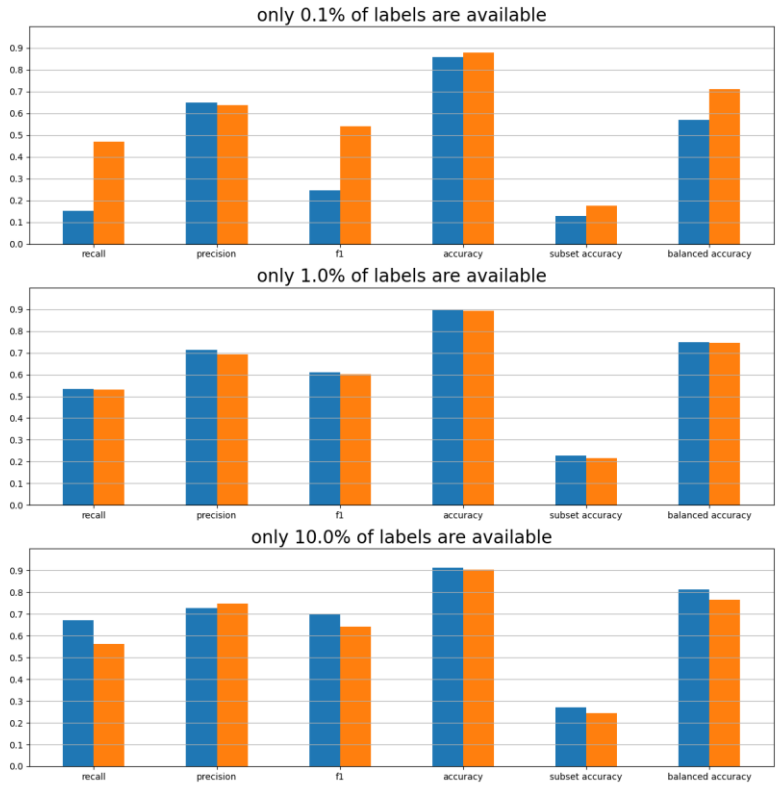


Figure 4: performance comparison under 0.1%, 1%, and 10% of label availability for the BEN dataset. The performance of the entirely trainable classifier is depicted in blue; the performance of the model composed of a SimCLR frozen trunk to which is added a linear trainable head (the "linear probe") is depicted in orange.

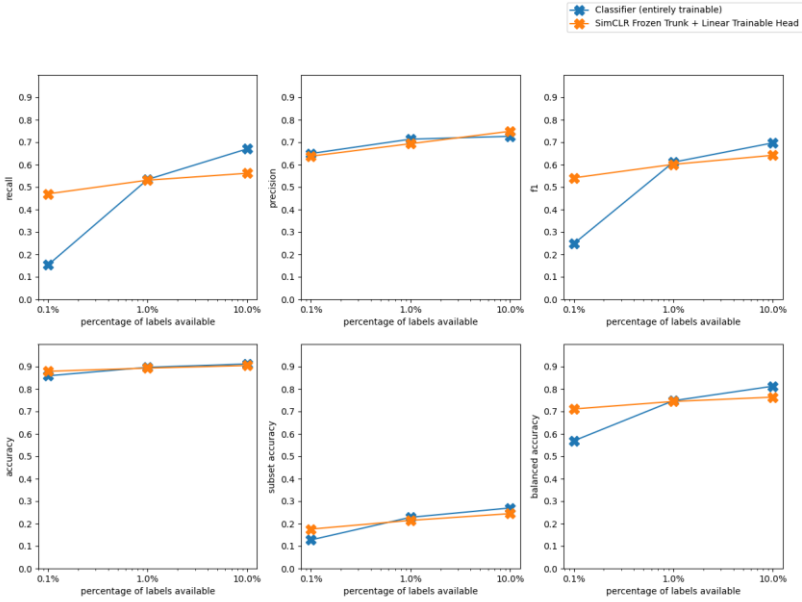


Figure 5: another view of the performance comparison under 0.1%, 1%, and 10% of label availability for the BEN dataset. The performance of the entirely trainable classifier is depicted in blue; the performance of the model composed of a SimCLR frozen trunk to which is added a linear trainable head (the "linear probe") is depicted in orange.

We start with the case in which the 0.1% data regime was used, corresponding to having only 270 labelled training samples for our supervised task. Under this severely low data availability level, our method consisting of learning a linear probe on top of a frozen architecture trained via self-supervision outperforms by a large margin the classifier. This gap is bridged when we move on to the 1% data scarcity regime. Furthermore, the SSL-based approach performs slightly under the classifier when provided with 10% of the total available labelled data. In an interpretation effort, we assume that, with the 27000 training samples available, the expressivity of the fully trained deep architecture of the classifier is higher than that of the weights of the single-layer probe that are being trained.

Overall, we observe that, indeed, under a limited labelled data regime, our approach not only outperforms the traditional supervised approach, but it does so by a significantly lighter training. More specifically, when using only 270 training samples, our approach achieves an above 0.50 F1 score, compared to an F1 of ~0.25 of the baseline, while our approach trains a single layer as opposed to the 101 layers of the baseline classifier. The extent up to which this reduction of data can continue being effective is a matter of experimental checking without theoretical guarantees. We expect however that as we reduce the percentage of labelled data further, we will start to witness diminishing returns, as 270 labels is already quite low when one considers the task at hand.

We describe the hyper-parameter used in Table 1.

Table 1: hyper-parameters used to train the SimCLR-inspired model via contrastive self-supervised learning.

Hyper-parameter	Value
Batch size	128
Gradient accumulation steps	8
Learning rate	0.0003
Layer-wise adaptive rate scaling (LARS)	yes

Weight decay coefficient	0.0001
Learning rate scheduler	Cosine annealing without warm restarts
Preload with weight trained on ImageNet	Yes
Penultimate layer size (fully connected)	128
Last layer size (fully connected)	64
NTX loss temperature	0.07

2.3 Learned Compression (LC) models

2.3.1 Introduction to Learned Compression

In recent years, learned compression has emerged as a powerful alternative in data processing, particularly for data with spatial or temporal structures such as images, sounds, and videos. This data inherently contains patterns, enabling high compression rates with minimized quality degradation. Unlike the standard lossy compression methods, which are heavily reliant on manual engineering and iterative refinement over decades, learned compression presents a different paradigm. Traditional methods, resulting in renowned codecs like JPEG for images and H.26x and MPEG for videos, as well as MP3 and AAC for audio, demanded extensive engineering expertise and manual tailoring to achieve optimal results. In contrast, learned compression, a hallmark of machine learning's adaptability, reduces the necessity for such manual intervention. Instead of being crafted and refined over time, it leverages training datasets, allowing algorithms to adapt organically based on data patterns. This dynamic, data-driven approach facilitates the creation of models that are intrinsically suited to specific datasets. For contexts with substantial data volumes, like the EO4EU project focusing on EO data, learned compression stands out as an efficient tool that balances compression efficacy with data integrity, all while reducing the engineering overhead typically associated with traditional methods when being applied to a novel data structure from scratch.

The integration of Vector Quantized Variational Autoencoder (VQ-VAE)[31] offers a contemporary approach to learned compression. VQ-VAE combines the attributes of variational autoencoders, adept at capturing intricate data structures, with quantization to create a concise, discrete representation of input data. This approach facilitates the efficient compression of data with spatial or temporal structures, such as images, sounds, and videos. By enabling the algorithm to identify and utilize underlying patterns in the data, VQ-VAE-based compression methods can achieve competitive compression ratios while preserving reconstruction quality. The application of VQ-VAE in learned compression suggests potential enhancements in data storage and transmission methods, with implications for areas like multimedia processing and data analysis.

At its core, a VQ-VAE incorporates three primary components: encoding, quantizing, and losslessly compressing, as depicted in Figure 6. The autoencoder, central to the process, employs an encoder-decoder mechanism to translate input data into a latent space and then revert it to its original form. This latent space becomes the arena for the quantization step, where continuous encodings are translated into a finite set of discrete vectors, streamlining the representation for storage efficiency. The final component, lossless compression (not represented in the figure), capitalizes on the quantized representation, further condensing the data by leveraging patterns within the discrete

[31] Van Den Oord, Aaron, and Oriol Vinyals. "Neural discrete representation learning." *Advances in neural information processing systems* 30 (2017).

vectors, reminiscent of traditional lossless compression algorithms like zip and tar.gz. The integrated function of these elements in VQ-VAE facilitates efficient data compression while preserving quality, and the design permits the model to identify complex structures in the data, making it adaptable across diverse compression scenarios.

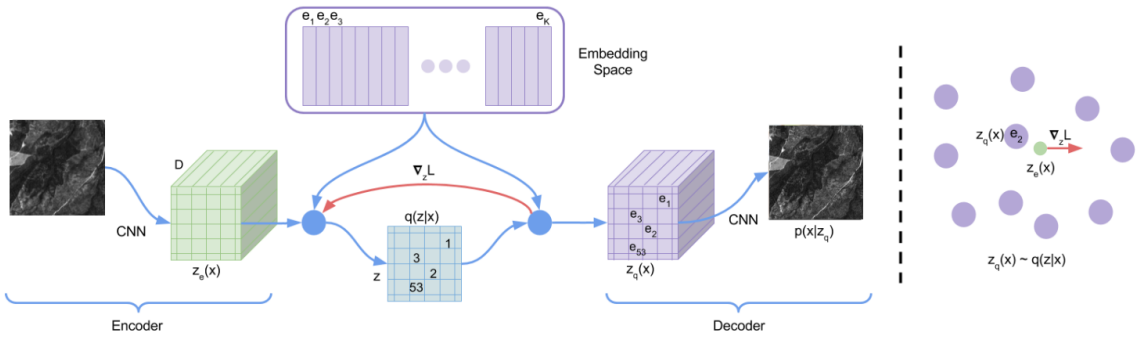


Figure 6: VQ-VAE
Modified image from the original paper. The original image is encoded using an encoder. The encoded representation is then discretized using a learned codebook of vectors. Finally, to ensure successful decompression, we learned a decoder that can recover the original data.

Encoder/Decoder:
When considering encoder-decoder pairs for temporal, spatial, and spatiotemporal data, the Convolutional Neural Network (CNN) emerges as a widely accepted choice. CNNs offer cost-effectiveness and a pronounced ability to discern patterns. Notably, they cater to the intrinsic characteristic of such data types: translation equivariance. This means that when a CNN receives translated data, the output similarly reflects this translation.

Quantization is the next phase, with a latent image as the foundational input. Here, the process occurs along the latent dimension, and each pixel in the latent space gets represented by a unique code. It's essential to highlight that the latent image's resolution might not match the original. Often, this latent representation undergoes downscaling, enhancing the model's interpretative prowess regarding data relationships. The 'dsf'(downscaling factor) determines the degree of this downscaling. Without downscaling (when $dsf = 1$), quantization in the latent space resembles colour indexing, with each colour uniquely identified. In contrast, a larger dsf allows for the indexing of broader data segments.

Moving away from traditional max pooling, this method incorporates a convolutional neural network characterized by its stride. For clarity, a convolutional layer with a stride of 2, as it traverses the primary data, accounts for every other data point, leading to downscaling by a factor of 2. Significantly, each output pixel draws from a broader area than the dsf might imply. This broader region, known as the "receptive field" in academic contexts, means that when downscaling is reversed during decoding, data accessed extends beyond just a single code. This ensures that even if a code undergoes modification, the change remains subtle and consistent with the dataset's overall structure. This approach suggests a potential reduction in artifacts within the learned compression, a topic we will detail further in a subsequent section. To counteract potential data loss from downscaling, the latent space's dimension is set larger than the original image's, ensuring comprehensive data capture.

Given an initial image with dimensions W x H x B — where W stands for width, H represents height, and B indicates channels—the encoded version takes on dimensions $\frac{W}{dsf}$x$\frac{H}{dsf}$x D. Here, D signifies the latent space's dimension. While the original image spans a spatial surface of WH, the encoded image covers $\frac{WH}{dsf^2}$.

Quantization:

The quantization phase aims to transform a real vector into a discrete code, making it storage-ready. This transformation is inherently lossy. Central to quantization is clustering. Data points in the same cluster are denoted by an index and reconstructed from the cluster center. Information about the exact position within the cluster is what is typically lost during this step. Two dominant clustering approaches are in use: the traditional K-Means clustering and a gradient-based algorithm. The K-Means method alternates between assigning data points to clusters and updating cluster centers. On the other hand, the gradient-based technique replaces the K-Means update rule with gradient descent. All cluster centers are collectively termed the codebook. As a result, the clustering algorithm outputs both the index of the assigned cluster and the value associated with a given data point.

Considering the worst-case, the bit requirement (b) to store $\#C$ clusters is:

$$b = \lceil \log_2(\#C) \rceil$$

For illustration:

- With 1 cluster, 0 bits are sufficient since there's only one category.
- 2 clusters necessitate 1 bit for differentiation (0 or 1).
- 3 clusters require 2 bits for three possible categories (00, 01, or 10), leaving the 11 code unused.
- 4 clusters also demand 2 bits, using codes 00, 01, 10, and 11.

Expressing this in terms of b :

$$\#C = 2^b$$

This reveals that as b bits of information are stored, the requisite cluster count escalates exponentially. This exponential growth directly influences memory usage during clustering. To counteract this exponential growth, Residual Vector Quantization (RVQ) [32] was introduced. RVQ curtails the complexity from exponential to linear relative to the codebook size by employing multiple codebooks. For instance, with two codebooks and a vector z to be quantized, the initial step quantizes z to get cluster z_0 with index c_0 using the first codebook. Then, the residual $r = z - z_0$ is determined, and the process is repeated, obtaining z_1 and c_1 from the second codebook. The final quantized vector and index are $z_0 + z_1$ and the concatenated c_0 and c_1, respectively. This method streamlines the complexity and memory demands of the quantization phase. The initial clustering sets a global position, whereas the subsequent clustering happens within the local cluster. Aggregation forms clusters, with an assumption of consistent structures within. Given $\#C_l$ as the cluster count per level, the total cluster count $\#C$ after N layers is:

$$\#C = \#C_l{}^N$$

This equation can determine the clusters per level ($\#C\_l$) required for a desired bit count b, as:

$$\#C_l = 2^{\frac{b}{N}}$$

An enhancement to RVQ is the integration of a variable codebook count. During training, for each sample, the number of codebooks (N) is chosen randomly within a predefined range. This adaptive approach ensures model versatility to various codebook counts.

A standout advantage of variable codebook counts is adaptability, allowing theapplication of a single model across different codebook quantities. For EO4EU, this adaptability is essential, reducing the models to be trained for distinct compression rates. Experiments presented in the subsequent section will demonstrate that this versatility has a minimal impact on model quality.

[32] Zeghidour, Neil, et al. "Soundstream: An end-to-end neural audio codec." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 30 (2021): 495-507.

Lossless compression:

The concluding compression phase uses a standard lossless encoding algorithm on the cluster index numbers. As posited by Shannon's source coding theorem, the efficacy of an ideal lossless compression algorithm can be bounded by the entropy derived from the probability distribution of the cluster indexes. The calculations from the prior sections present a worst-case scenario, which is most pertinent in situations where the data is uniformly random, leading to maximized entropy. Subsequently, in the evaluation section, we adhere to standard practices in the literature, noting the theoretical compression ratio. This is especially relevant considering that multiple optimal algorithms can achieve this ratio when the index distribution is well-defined.

2.3.2 Data type selection and data preparation

Much like the discussions in Section 2.2.2 on data type selection for SSL models, for learned compression we chose Sentinel-2 multispectral images as our guiding data. We leveraged the auxiliary code blocks developed for SSL. Unlike SSL where parts of the data were removed, we preserved all data. In a compression context, factors such as high cloud coverage of satellite images do not compromise the task accuracy. The aim is to faithfully reconstruct the original image, independent of its content. Otherwise, the data preparation mirrors the SSL approach, as detailed in Section 2.2.2. The training dataset comprises roughly 413k samples, while the test set contains about 88k samples.

When it comes to the training and development of learnt compression models on other data sources the setting is in a sense even simpler than the one in SSL. One needs to only adjust the input layers to the input data structures and obviously there are no augmentations to consider. We will provide compression models for at least one more data source used in the use cases. We see a considerable opportunity, as well as challenge, for learned compression on ECMWF CAMS datasets which have a three-dimensional, instead of two-dimensional structure. We can practically adopt the same approach as in the Sentinel-2 case where we simply replace the 2D convolutions over the input data with 3D convolutions. ECMWF has published a nature paper where they release a full resolution dataset without internal compression[33]. The CAMS dataset[34] presented in the paper is 153 MB for 1 level and 21 GB for the 137 levels for the whole world on 109 variables.

Having a model architecture that treats both 2D and 3D data should enable the treatment of the majority of grid like data structures. For example, ERA5 Land has the same structure as Sentinel 2 data and our model can be retrained to support these data sources.

2.3.3 Model Evaluation protocol

In evaluating learned compression models, the challenges differ from those in a supervised approach. In the latter, a single performance metric can be sufficient to compare different models. However, for learned compression, there's a need to balance two essential objectives: the degree of compression and the accuracy of the reconstructed output. A compression model is said to Pareto dominate another if it excels in both these criteria. In cases where no model has clear Pareto dominance, the

[33] Klöwer, M., Razinger, M., Dominguez, J.J. *et al.* Compressing atmospheric data into its real information content. *Nat Comput Sci* **1**, 713–724 (2021). https://doi.org/10.1038/s43588-021-00156-2

[34] *CAMS Forecast Experiment using GRIB IEEE Data Encoding* (CAMS, 2021); https://doi.org/10.21957/56GH-9Y86

choice becomes subjective, influenced by the specific problem, its demands, and the preferences of the researcher or designer. Our goal is to juxtapose our solutions with a relevant baseline, thereby highlighting the interplay between compression and fidelity.

To visualize this trade-off, we plot it on a two-dimensional graph. This representation provides a straightforward understanding of the tension between compression level and reconstruction accuracy. As a benchmark, we use the JPEG algorithm, interpreting the images from each band as monochromatic, single-channel visuals. To traverse this landscape of trade-offs, we select diverse compression ratios, noting the corresponding fidelity (quantified by compression error) alongside.

The compression degree is gauged using the metric "bit per pixel" (bpp). This metric is derived by dividing the total byte size of the image by its pixel count. For JPEG, we consider the file size. For learned compression, we opt for entropy to ensure our metrics aren't influenced by the choice of lossless compressor.

The maximal compression ratio for learned compression is found by determining the bits needed to represent a single code, based on the number of clusters per layer ($\#C_l$) and the layer count (N):

$$b = \lceil N \log_2(\#C_l) \rceil$$

The entire code count is:

$$\frac{WH}{dsf^2}$$

Subsequently, the bit per pixel (bpp) is:

$$bpp = \frac{N \log_2(\#C_l)}{dsf^2}$$

For measuring errors, we deploy the root mean square error (RMS) as a percentage. This error is normalized to ensure values range between 0% and 100%.

2.3.4 Model Training

In our initial investigations, we utilized a variant of VQ-VAE[35], aiming to limit codebook usage to boost performance. However, early tests revealed a tendency for the model to prioritize reducing the codebook size, sometimes using only a few of its elements. This phenomenon, known as "degenerate outcomes," necessitated an adjustment. We briefly experimented with a modified regularizer, focusing solely on reconstruction error. This quickly underscored the scalability concerns that had originally led to the introduction of RVQ.

During this phase, we also found Batch Normalization to be particularly batch size-sensitive, prompting its replacement with Layer Normalization. Furthermore, we settled on a down-sampling factor (dsf) of 2, striking a balance between memory efficiency and the risk of over-smoothing from excessive downscaling.

Figure 7 visually depicts the encoder's intricate architecture, which is divided into three main segments:

Optional Downscaling: Situated at both the start and end of the architecture, this section uses convolution with a stride of 2 for downscaling when needed, and standard convolution otherwise. For instance, with a dsf of 2, only the initial downscaling is activated; for a dsf of 4, both are engaged.

[35] Gregorová, Magda, Marc Desaules, and Alexandros Kalousis. "Learned transform compression with optimized entropy encoding." *arXiv preprint arXiv:2104.03305* (2021).

Convolutional Core: This central segment boasts two main blocks, each housing three sub-blocks. These sub-blocks each consist of two convolutional layers, augmented by a ReLU non-linearity and layer normalization. This arrangement amasses a total of twelve convolutional layers throughout.
Residual Connections: To bolster information retention and continuity, residual skip connections are introduced between each primary block at every hierarchical stage. This structural decision ensures that data is consistently maintained across the hierarchy, allowing the model to selectively infuse data when deemed essential.

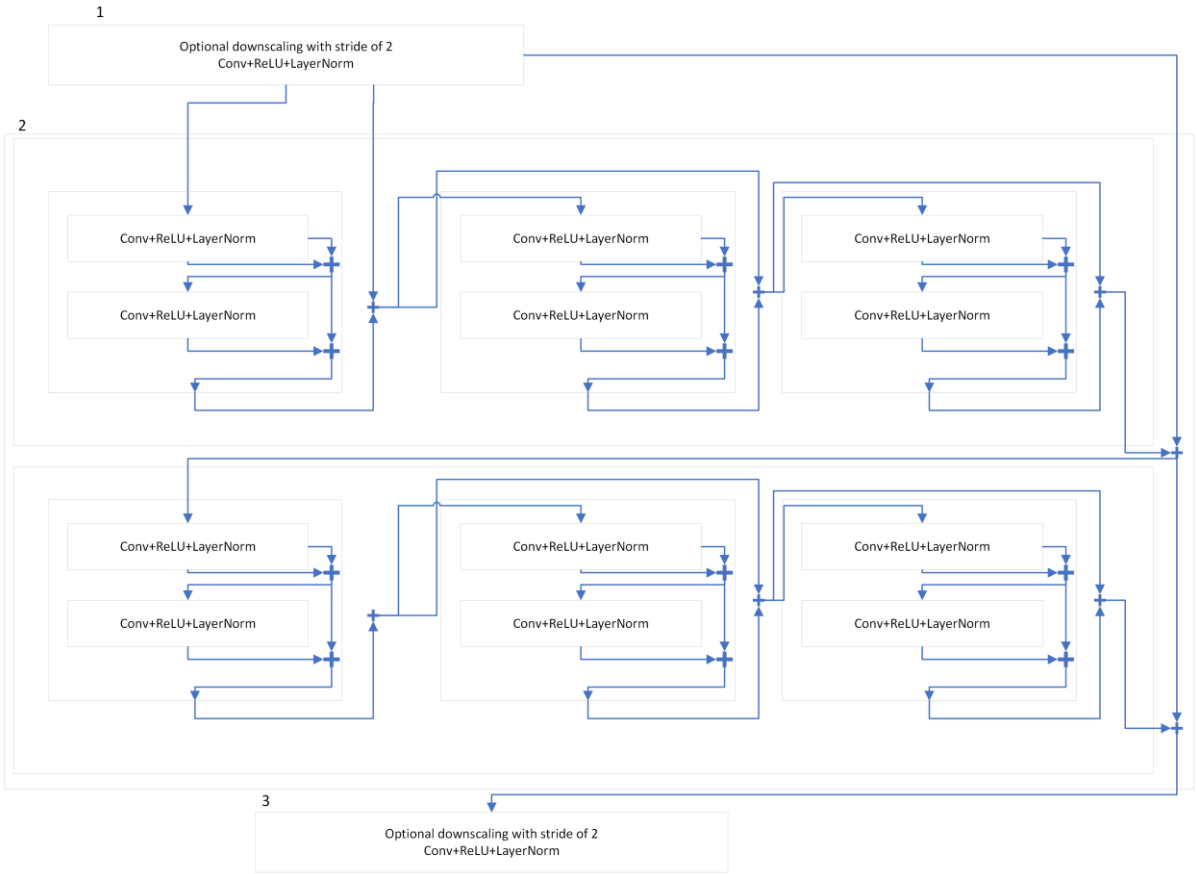


Figure 7: Encoder Architecture. The black numbers refer to the architecture description. The "+" node denotes addition operations. The blue arrows indicate data flow. The boxes denote component groupings.

Decoding is relatively straightforward; we use a mirrored version of the encoder architecture. This symmetry involves transposed convolutions, effectively backtracking through the encoding steps to recreate the initial data.

The models are trained using Adam optimizer with a fixed learning rate.

During training, we encountered two primary challenges. First, GPU memory restrictions were a constraint. This was accentuated by the Sentinel data's >10 channels—compared to the typical 3 in RGB images—and the use of expansive image patches. The second challenge stemmed from the sheer size of the dataset. This necessitated optimized I/O during training to ensure timely completion.

2.3.5 Model finetuning and evaluation

The key hyperparameters requiring fine-tuning include:

Downscaling Factor (dsf): Represented by the number of layers with a stride of 2, the dsf determines the reduction level applied to the original image (1x, 2x, or 4x). It directly influences the dimensions of the spatial latent space and the size of a single latent code's receptive field. Intensified downsampling requires the model to encapsulate a larger spatial region within a single code, potentially leading to finer detail loss. Conversely, limited downsampling retains more details but compromises compression efficacy. Although we primarily employed a 2x factor, we also ventured into analyses with factors of 1x and 4x.
Latent Space Channels: The number of channels in the latent space is pivotal. Our selection aimed to strike a balance—choosing a figure substantial enough to avoid overfitting, considering our dataset's size, while also being mindful of memory constraints. We settled on 128 channels. Notably, this parameter doesn't have a direct bearing on compression. Hence, our detailed exploration was limited. Furthermore, to ensure that the latent space's size wasn't restrictive, we trained an autoencoder without quantization, confirming that the latent space wasn't a bottleneck.
Codebook Configuration: This involves the number of codebooks (N) and the size of each codebook($\#C_l$). Directly impacting the model's compression and quality, the effective codebook size is derived as $\#C_l{}^N$. Our typical configuration utilized codebooks of size 10,000 and incorporated up to 8 codebooks.
In essence, the primary determinants of compression are the dsf and the codebook's configuration—both its size and quantity. Other parameters, while influential, are primarily constrained by GPU memory capacities, especially when handling large datasets, rather than influencing compression directly.

The experiments conducted are visualized in Figure 8. The x-axis represents compression in bpp, while the y-axis displays the RMS in percentage. Here's a breakdown of the conducted experiments:

- **Dynamic Number of Codebooks (Blue Dots)**: A model was trained using a dynamic range of codebooks, varying from 1 to 4, over the course of a week. The outcomes of this training are represented by blue dots on the graph.
- **Fine-Tuning with Fixed Number of Codebooks (Blue Crosses)**: Following the initial training, each of the 4 models was further fine-tuned, with a fixed number of codebooks, for an additional four days. These results are indicated by blue crosses in the figure.
- **Evaluation without Quantization (Continuous Blue Line):** The performance of the model, when evaluated without quantization, is depicted as a continuous blue line on the graph.
- **Continuous Model without Quantization (Black Line):** A continuous model, devoid of quantization, was trained over a span of 4 days. It leveraged weights from the preceding dynamic model. The resultant performance is shown as a black line, serving as a reference point or lower bound for the given training time and network architecture.
- **Eight Dynamic Number of Codebooks (Orange Dots)**: A model was trained over four days with eight dynamic codebooks. This model was initialized using the weights from the earlier continuous solution. The outcomes of this model are symbolized by orange dots on the figure.
- **Evaluation of the 8-Codebook Model without Quantization (Continuous Orange Line)**: The same model, when assessed without quantization, is represented by a continuous orange line on the chart.
- **JPEG Performance**: The efficiency of the JPEG standard was also assessed by modulating the quality parameter in the compression.

The results of these various experiments offer insights into the capabilities and potential enhancements of the models, as well as a comparative analysis with traditional compression methods like JPEG.

Based on the conducted experiments and the data visualized in Figure 8, the following conclusions can be drawn:

- **Efficacy of Dynamic Number of Codebooks:** The use of a dynamic number of codebooks seems to be effective with a marginal performance cost. This is evident from the proximity of results between models depicted by the blue dot (dynamic codebook model) and blue cross (fine-tuned fixed codebook model).
- **Performance of Autoencoders Without Quantization:** For models trained with quantization, the autoencoder's performance without quantization tends to align with the performance achieved using the highest number of codebooks during training.
- **Potential for Autoencoder Performance:** The autoencoder's potential hasn't been fully tapped, as suggested by the superior performance of the model trained without quantization (represented by the black line).
- **Saturation of Performance Improvement**: As the bpp escalates, the performance enhancement of the model plateaus post-20 bpp. It's speculated that this stems from the inherent limitations of the learned compression algorithm. When trained for a constrained period using a finite dataset, the algorithm possesses intrinsic errors. In comparison, traditional algorithms like JPEG, when set to maximum quality, are mathematically designed to retrieve the original data, whereas the machine learning counterparts aren't.
- **Model Accuracy:** A significant majority of the models tested showcased errors below the 2% threshold.

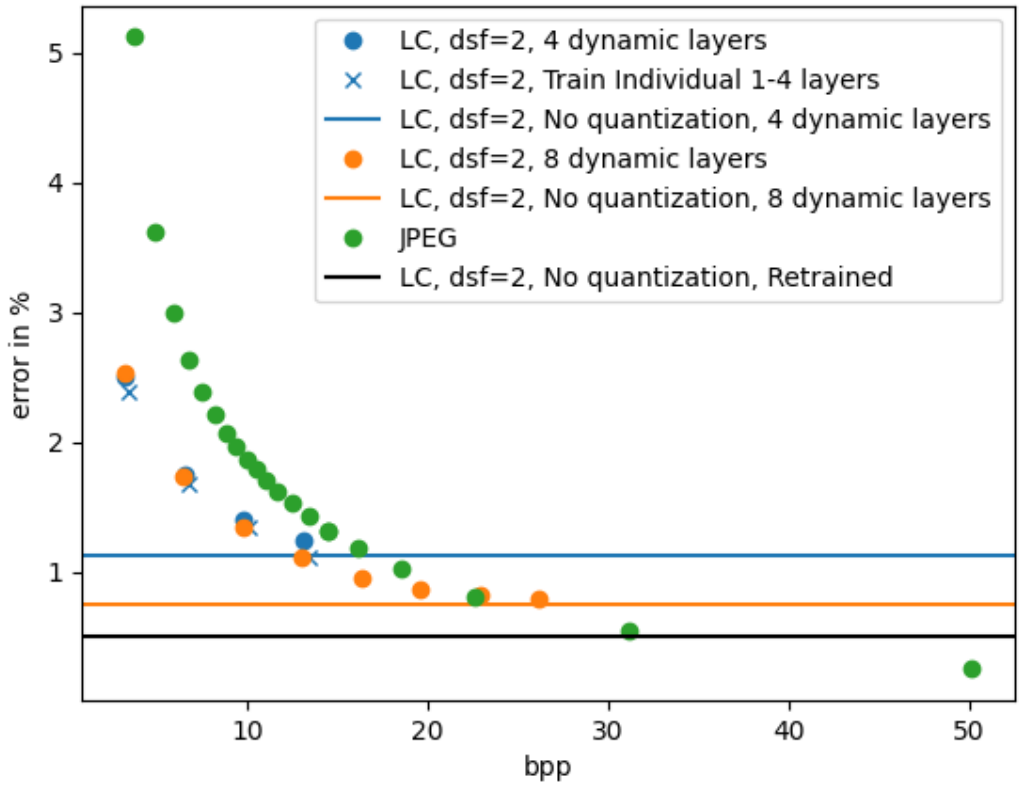


Figure 8: Reconstruction quality versus compression rate.
On the x axis we show the compression rate expressed as bit per pixel, on the y axis we show the error expressed in

percentage. Each dot consists of a trained model and its performance. Line represent performance of non-quantized models. Data raw bpp is 120.

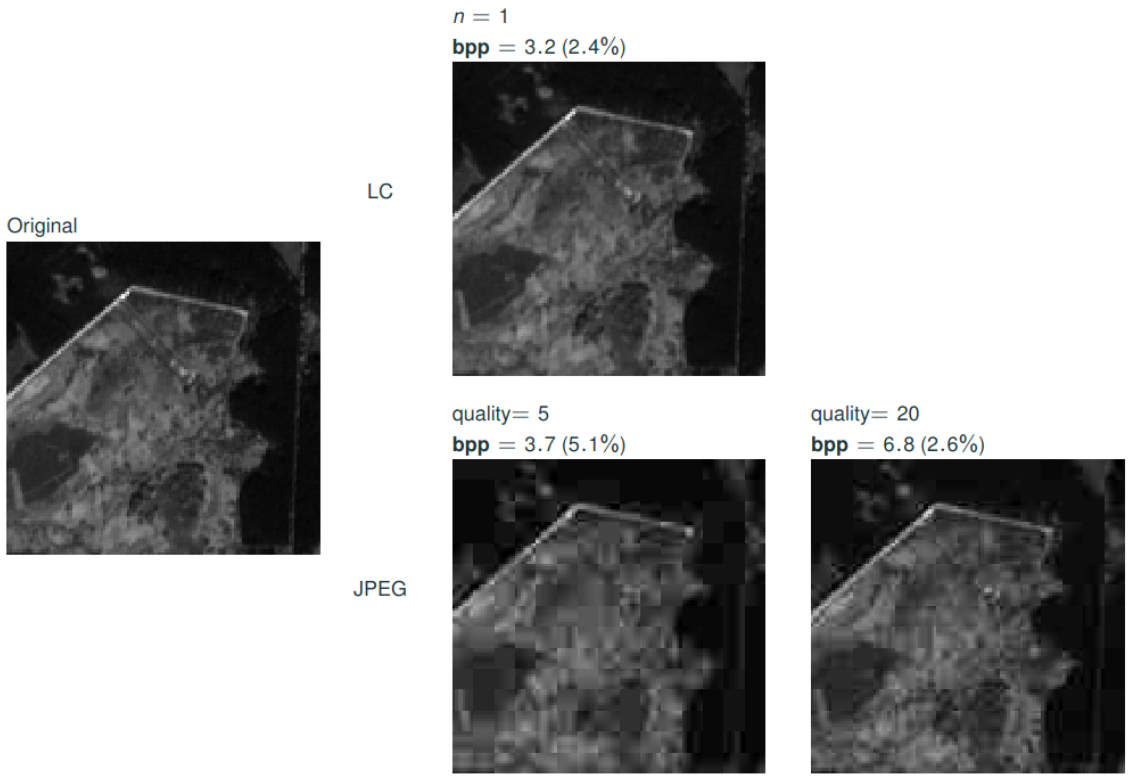


Figure 9: Visual comparison of a single instance of the first band between the learned compression model with a single quantization layer and JPEG. The first JPEG image has similar bpp whereas the second JPEG image has similar error.

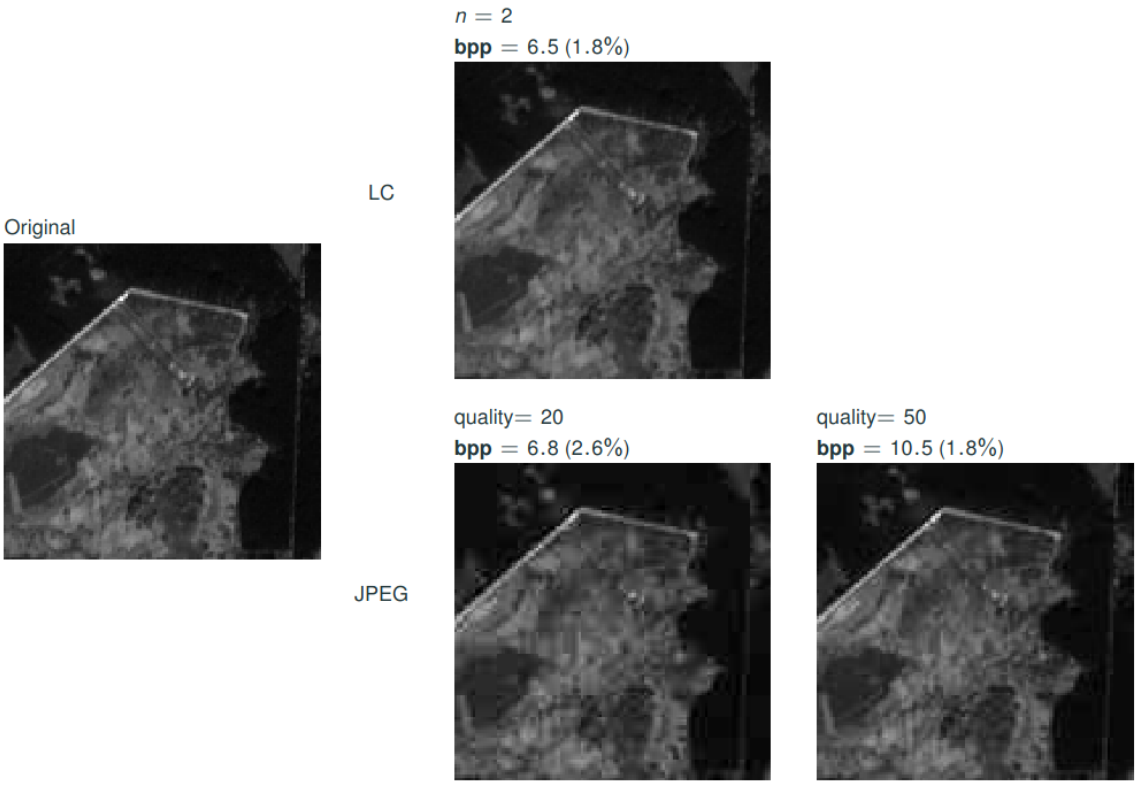


Figure 10: Visual comparison of a single instance of the first band between the learned compression model with two quantization layer and JPEG. The first JPEG image has similar bpp whereas the second JPEG image has similar error.

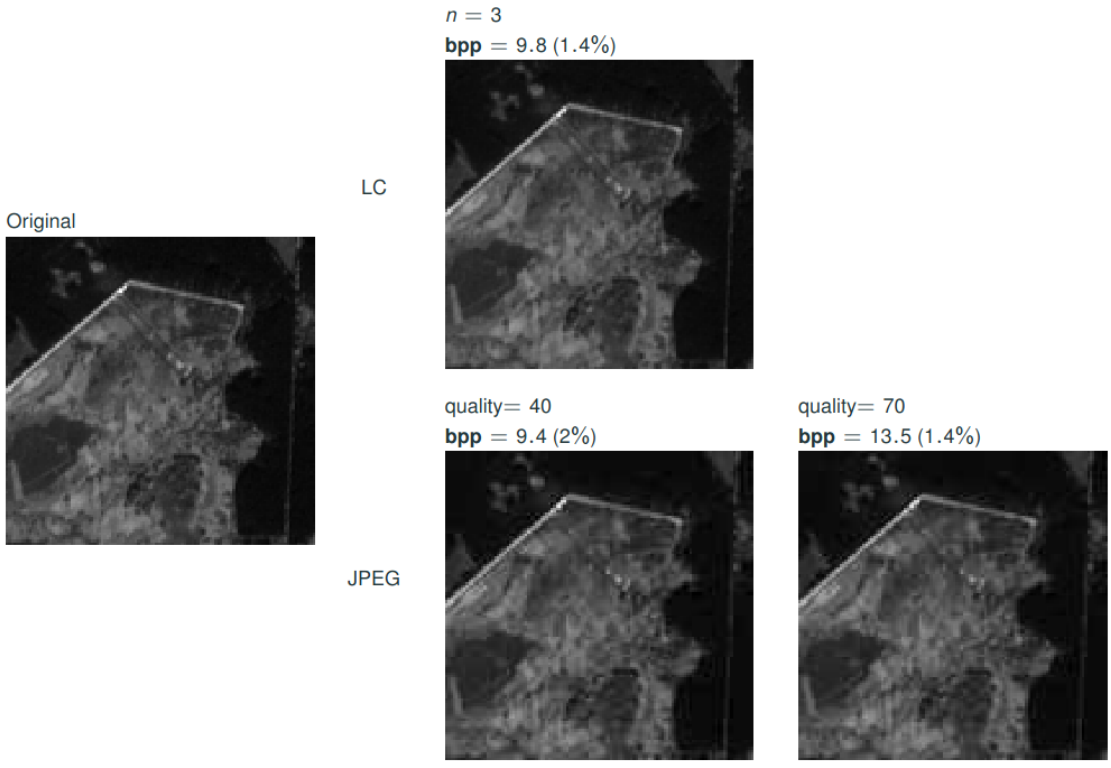


Figure 11: Visual comparison of a single instance of the first band between the learned compression model with three quantization layer and JPEG. The first JPEG image has similar bpp whereas the second JPEG image has similar error.

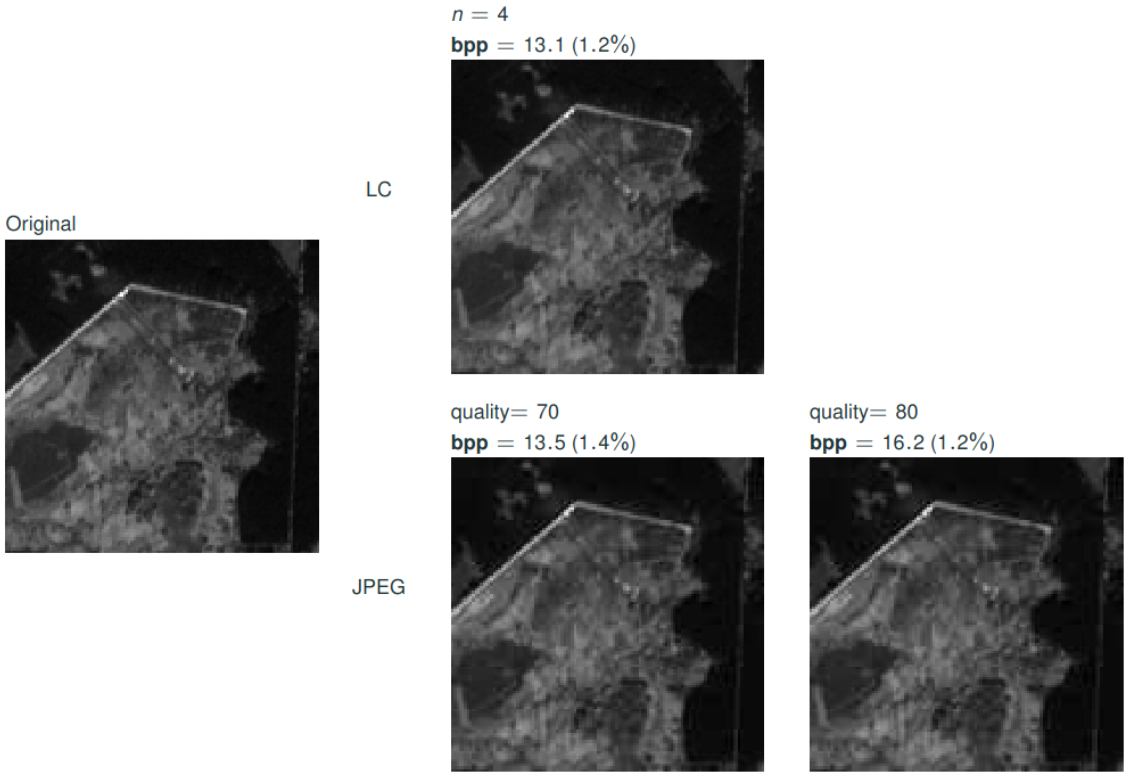


Figure 12: Visual comparison of a single instance of the first band between the learned compression model with four quantization layer and JPEG. The first JPEG image has similar bpp whereas the second JPEG image has similar error.

The purely quantitative assessments, though informative, offer only a partial understanding of the overall performance. A qualitative examination of visual quality provides another layer of insight, as evidenced when comparing images compressed using JPEG and learned compression methods.

Figure 8 through 12 display this comparison, focusing on the use of 1 to 4 layers of quantization. A noticeable characteristic of JPEG compression is the presence of blocking artifacts; these manifest as 8 by 8 pixel blocks which directly correspond to the block size utilized by the JPEG algorithm. In stark contrast, images compressed via the learned method are devoid of such artifacts.

This superior visual consistency in learned compression is likely attributable to its holistic approach to image compression. Rather than segmenting and processing the image in discrete blocks, the learned compression processes the entire image as an integrated entity. This method ensures smooth transitions and consistent patterns across neighboring pixels, eliminating the blocking artifacts that are evident in JPEG.

This research demonstrates the efficacy of learned compression techniques. Not only do these methods achieve efficient image compression, but they also preserve the perceptual quality of images, making them a competitive alternative to traditional methods like JPEG in terms of compression and visual fidelity.

2.3.6 Quantifying the effect of compression with respect to downstream tasks:

To assess the impact of compression on machine learning tasks, we evaluated the binary cross-entropy error of a baseline classifier when applied to compressed versus original data. We employed T-tests to compare the error rates of compressed versus original data to determine if the compression resulted in a significantly different error. Our findings indicate that there is no substantial difference in error across all models, with the exception of the lowest compression setting, which utilizes only one out of eight codebooks. Interestingly, in this setting, the compressed data yielded a slightly better result, reflected in a p-value of 0.02. However, considering the marginal nature of this improvement and the higher p-values observed in other settings, the results do not appear to be significant.

We also computed the symmetric KL divergence for every instance for each label individually and created their histogram. The histogram follows a distribution with a very long tail, with the majority of the probability mass near 0. This is visualized in Figure 13.
Finally, to quantify the effect on class assignment, we categorized the label into four categories: Surely No, Maybe No, Maybe Yes, Surely Yes, with boundaries at probabilities of 25%, 50%, and 75%. We created the contingency matrix of these categories and found that, as expected, the majority are near the diagonal. This shows that there are only a few changes in classification due to compression. We visualize one of these results in Figure 14.

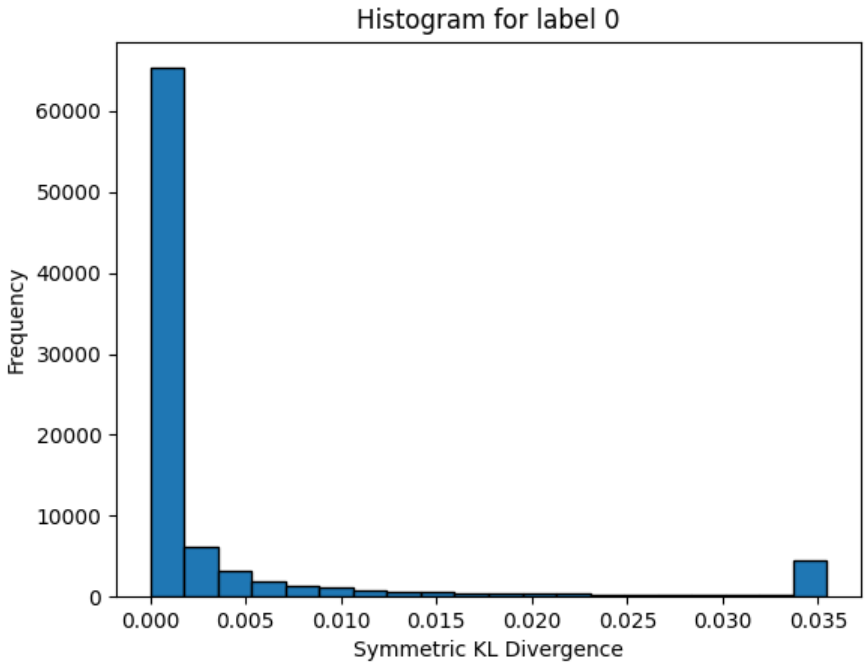


Figure 13: Histogram of the first label of the Symmetric KL Divergence of the most compressed model. The last bin is unbounded and contain 5 percents of the data.

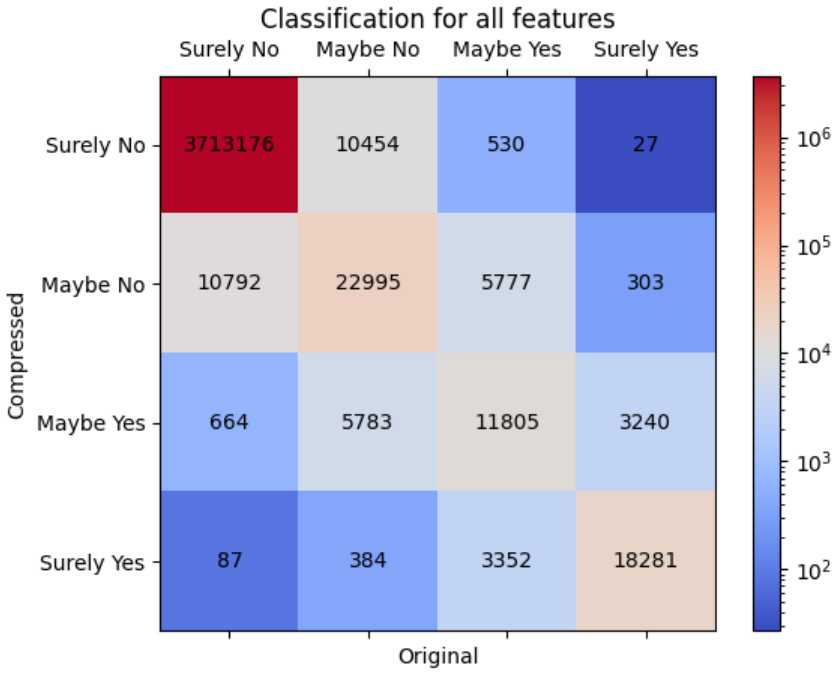


Figure 14: Contingency matrix for all labels showing the miss classification due to compression. This is done here with the most compressed model.

2.4 Model Documentation

Effective model documentation is at the heart of ensuring the transparency, reproducibility, and usability of the Machine Learning (ML) models developed within the EO4EU project. We recognize the significance of providing comprehensive documentation that empowers users to harness the potential of these models. To achieve this goal, we employ a multi-faceted approach to documentation, leveraging various methods to offer a clear guidance to the users of the models, regarding their operation.

One of the cornerstones of ensuring thorough model documentation and unambiguous guidelines for their proper usage is by providing accompanying metadata files that offer detailed information about both the syntax and semantics of the models. This metadata not only clarifies how the data fed to and returned by the models are structured but also provides a comprehensive view of their underlying semantics, aiding users in understanding what constitutes a valid and reasonable use of the models. For instance, the dimensionality of the input tensor that trained models expect is important for the validity of the input. Moreover, apart from reporting for dimensionality of the input tensor, we also document the type of data that the models expect, with all relevant details. Apart from reporting the number of channels of the input tensor, we also report the type of data (for instance, Sentinel-2 satellite images) and of each channel (the order of the bands of Sentinel-2 comprising said tensor). In this way, we ensure not only the structural validity of the data that will be fed to the models, but their semantic relevance as well.

Furthermore, our code development practices prioritize self-explanatory code, ensuring that the logic and operations of the models are clear and self-explanatory, as modern code development requires. This approach not only enhances the accessibility of the codebase but also promotes collaboration among developers. Naturally, where necessary, we incorporate comments within the code to provide

additional clarity, making it easier for users to navigate and comprehend the implementation details. This combination of clear code structure and thoughtful commentary ensures that the models can be seamlessly understood and extended not only by our consortium's collaborators but also by the external interested user.

A pivotal element assisting our documentation strategy is the provision of real-world usage examples within the EO4EU platform. By showcasing practical applications of the models, users gain insights into how the models can be integrated into their specific projects. These examples serve as valuable templates, helping users adapt the models to their unique needs.

Lastly, comprehensive and extensive written descriptions, in line with the content of the current deliverable, offer a contextual overview of the model development process and their intended use. These descriptions are set in wider context, providing an overview of the project's objectives, the methodologies employed, and the intended use cases. This contextual backdrop offers users a holistic understanding of the models' purpose and relevance within the EO4EU project and beyond.

In essence, our documentation approach combines metadata, code clarity, practical examples, and written descriptions to ensure that users have access to a rich array of resources that enable them to effectively utilise and integrate the ML models into their own workflows.

End users will gain access to model documentation through multiple channels. The platform itself will offer documentation via GUI elements and links to more comprehensive resources. Additionally, we are exploring the possibility of incorporating error checking and diagnostic features to ensure users input syntactically correct data.

Furthermore, model documentation and all models cleared for public release (contingent upon use case owner approval) will be accessible via the Zenodo platform within the EO4EU community, and more broadly, on the EOSC platform. We offer all essential resources necessary to replicate everything that revolves around inference for any given model, this includes code and data that can be used to train the models from scratch, examples of model deployment, calls to the models on the EO4EU inference server. These scripts also closely replicate all inference functionality provided within the EO4EU platform, from data retrieval, data preprocessing and data shaping for inference, to inference and exploitation of inference results in downstream applications. Utilizing the models provided on the EO4EU platform allows users to take full advantage of all available functionalities, offering a more seamless experience compared to using the standalone versions provided in zenodo.

More concretely, for every model that has been made available in the platform the documentation provided encompasses the following:

- **Model Documentation and Development**: Comprehensive written documentation detailing the models and their development process.
- **Sample Data for Inference**: This includes sample data for inference purposes, along with a script to retrieve this sample data from its source.
- **Sample Data for Training**: Similar to the inference sample data, this includes sample data for training, accompanied by a script for sourcing this data.
- **Model Deployment Instructions**: Detailed instructions on how to deploy an inference server locally, along with the full model.
- **Demonstration Notebooks**: Notebooks demonstrating the use of the inference server with data, showcasing practical applications and examples.
- **Training Script**: A script designed for training a model from scratch, enabling users to understand and replicate the training process.

- **Machine Learning Component Code**: The code for all machine learning components available on the platform, providing a deep dive into the technical underpinnings and integration.

The documentation can be found at the following URL: https://zenodo.org/doi/10.5281/zenodo.10814032.

3 Integration and Testing of ML Models

3.1 Methodology

We build upon the global architecture of the EO4EU platform in which the different components communicate through the message bus. The ML component has the following three subcomponents:

1. ML Controller:
   This component work in coordination with the EO4EU platform orchestrator to spin up the necessary resources for a new workflow.
2. Wrapper:
   This component serves as the entry point of an ML workflow step. It is responsible for reading the user query, accessing the workflow configuration, obtaining the necessary data, communicating with the inference server to execute the query, and finally storing the result.
3. Inference Server:
   This component is a specialized FaaS dedicated solely to inferring ML models. It receives a query with embedded data and responds with the result.

A summary of the architecture is provided in Figure 15 together with the communications between the components.

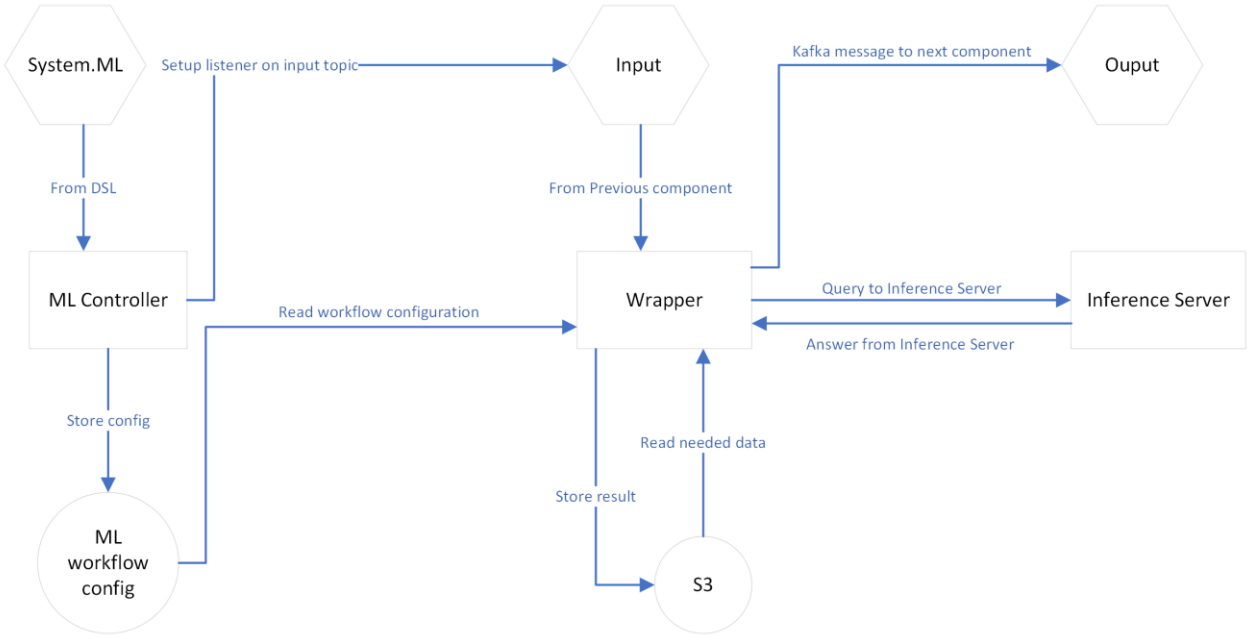


Figure 15: Architecture of the integration. Square boxes represent components. Hexagons indicate a Kafka topic, and circles indicate storage

3.2 Integration Environment Setup

3.2.1 Infrastructure

Both the Controller and Wrapper components function as Python WSGI web applications. To integrate with the communication technology of EO4EU (specifically Kafka), we employ KNative resources. These resources facilitate the transformation of Kafka messages into HTTP requests for our web applications used for both our Controller and Wrapper. More precisely, the KafkaSource resource enables listening to a specific topic and subsequently sending an HTTP request containing the original message to a Service. The ML Controller component listens for messages in the 'system.ML' topic sent from the DSL. Upon receiving a message from the DSL, the controller stores the configuration and creates a new KafkaSource to monitor the new workflow topic. The Wrapper, in turn, receives a message from the preceding component in the workflow. It then reads the configuration for the workflow, downloads the data, and utilizes the internal inference server to run an inference with the model on the data. Ultimately, the result is uploaded to S3 in a shared bucked for a give workflow. The path to these results is included in the message sent to the subsequent component.

The Wrapper component is present for every ML model and dataset. Its structure is as follows:

1. Read the configuration from storage and the query.
2. Load and parse data from S3, then perform the required preprocessing.
3. Call the inference server.
4. If necessary, transform the result into a specified format and store it on S3.
5. Send a message to the subsequent component with a link to where the result can be found.

It's worth noting that steps 2 and 4 vary based on the data format. While they are currently written manually, there are ongoing efforts within the consortium to generalize and simplify this type of boilerplate code.

The inference server is the only component that interacts directly with the ML models. We employ KServe as our inference server orchestrator. Before delving into KServe, let's discuss the concept of inference server. An inference server can be thought of as analogous to a web server, but instead of serving web pages, it serves ML models. The primary objective of an inference server is to offer high throughput to clients, typically via an API, often REST-based. These servers are also engineered to leverage accelerators such as GPUs and TPUs. Various inference servers exist, each supporting its own model format. We have opted for NVIDIA Triton, which utilizes the ONNX format. This choice was motivated by the fact that ONNX is a versatile format compatible with most ML frameworks. Models are organized within a directory structure, accompanied by a configuration file that describes the model's input/output specifications and a binary file containing the model itself. Note that, in most cases, an inference server does not need the original code of the model.

KServe simplifies the orchestration of inference servers by offering the following features:

1. A single resource (InferenceService) to configure and deploy an inference server.

2. Supports automatic scaling based on load, including scaling from zero (uses no resources if there is no load).
3. Manages routing and DNS.
4. Provides production-ready software on Kubernetes.

Currently, the inference server is used exclusively by the ML components. However, other partners could utilize it if they ever have the need.

In the context of the EO4EU project, we will organize the various inference servers based on similarities in data and use cases to simplify administration. Currently there is only a single inference server however in the future as the more model are deployed additional server will probably be added.

On the EO4EU platform, the ML components currently reside on a standalone Kubernetes server. This setup allows all communication to occur exclusively through the message bus (Kafka) or the inference server. Additionally, this configuration facilitates easy relocation of the ML components to another physical site if necessary

3.2.2 Data access

Data is accessed by the wrapper component using the protocol defined by NKUA.
The location and credentials for the data are sent by the DSL to the ML Controller, which then stores them for use in the wrapper. At present, data is stored in S3, with a single tar.gz file for each component. We have considered and experimented with using an uncompressed format instead. The responsibility for data storage and parsing is confined to a single component of the codebase, the wrapper, which simplifies modifications if necessary.

3.2.3 Validation and Testing of the ML component

Our approach to validation and testing is tiered across three distinct levels to ensure robustness and reliability:

1. **Model-Level Evaluation:**
    - At this foundational stage, every model undergoes rigorous testing, evaluation, and documentation before any consideration for integration into the platform.
    - As discussed in Section 2, every model is meticulously documented and evaluated. However, only models that exhibit superior performance are selected for deployment and presented to end-users.
    - While this level emphasizes the model's machine learning performance, it abstracts away the complexities that may arise due to platform integration.
2. **Backend Tools Validation:**
    - This level focuses on the backend transformations necessary for model deployment.
    - The transformations are built upon standardized, tried-and-tested machine learning formats such as ONNX, PyTorch, and Tensorflow. Each format is rigorously validated by their respective parent projects.
    - Deployments are executed on standard inference servers to ensure reliability and consistency.
3. **End-to-End Integration Testing:**
    - This comprehensive level aims to ascertain the seamless integration and functioning of custom codes within the platform.

- The primary goal is to confirm that input-output formats align with platform expectations and that no unexpected behaviors or discrepancies manifest.
- Specifically, we conduct two primary tests:
    1. A workflow involving SSL, culminating with a classifier, is initiated. The subsequent steps involve validating the accuracy of the file outputs for both SSL and the classifier.
    2. Another workflow launches with Compression, followed by Decompression. The main validation check here is to confirm the usability of the output as a discernible file format.
- Additionally, our team remains in constant dialogue with components that utilize our outputs and supply inputs to us. This continuous communication ensures mutual compatibility and streamlined operations.

In summation, our multilayered validation and testing strategy, ranging from individual model evaluations to end-to-end integration checks, endeavors to ensure the reliability, accuracy, and compatibility of our platform components.

3.3 Interaction with other EO4EU components

3.3.1 Fusion Engine

Component	**Fusion Engine**
Responsible partner	NKUA
Participant partners	MEEO, CMCC
Technical description	This component is responsible to preprocess the Data. Currently the transformation done by Fusion related to ML is the rescaling of different resolution of Sentinel-2 data to the same resolution.
Relation to the ML models	ML component work in close collaboration with fusion as they do the majority of Data Preprocessing needed for our components.

3.3.2 DSL Engine

Component	**DSL Engine**
Responsible partner	NKUA
Participant partners	
Technical description	The DSL Engine is a tool allowing to create a workflow using a GUI/CLI and execute the workflow.
Relation to the ML models	As the ML component is separate from the other component, the DSL communicate with the ML controller to orchestrate needed resources from the workflow. The DSL Engine team is also working with all other partners to ensure a compatible communication of all steps in the workflow including ML component.

3.3.3 AI ML Marketplace

Component	**AI ML Marketplace**

Responsible partner	EBOS
Participant partners	NKUA
Technical description	The ML Marketplace is discussed as a place where workflow and models can be stored and created by user using some kind of GUI. The exact scope of this component is still under active discussion with the Partners.
Relation to the ML models	As we are developing the core ML algorithms of the platform and working with the use cases partner having ML algorithms need. We will assist in giving information about ML models that are deployed and assist on how to extend ML steps with new models, wrappers and other ML related help the partner might need.

3.3.4 Other components

Every component that utilizes the machine learning results must adhere to a specified output format when reading our results.

3.4 Integration Test Results

At the conclusion of the online integration test in July, we successfully integrated four models that illustrate how the platform will be employed for ML.

To validate the integration, we experimented with two distinct combined workflows using the BEN datasets, with data pre-processed by Fusion.

First, we set up a workflow to test SSL. For this, we linked two ML components:

1. The feature extractor from SSL, which extracts features from the data.
2. The downstream classifier we trained, aimed at testing the performance of SSL on a downstream task.

In practice, this component is designed to be used in two ways:

1. Utilizing a downstream algorithm either created by the consortium or created by the user.
2. Extracting features with SSL and allowing the user to handle the training on their end.

We tested a workflow for learned compression that involved two steps:

1. Compression.
2. Decompression.

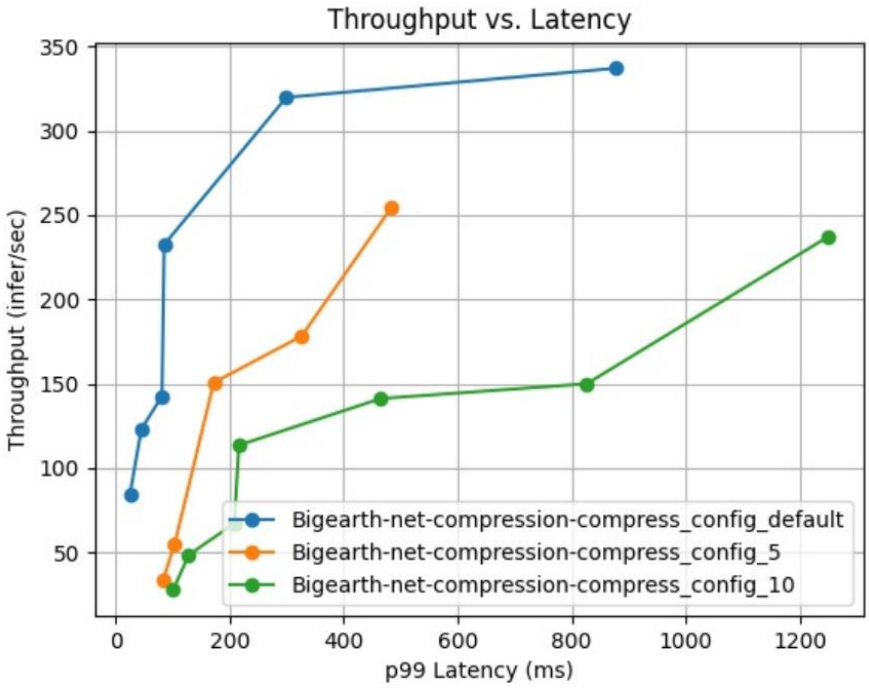
The output from this process should be indistinguishable to the user.

In practice, the output from the Compression step is intended for storage or transmission to another location where Decompression takes place. However, for our testing purposes, we executed both steps on the same cluster.

To motivate the infrastructure needs of having GPUs on the cloud for the inference server we evaluated the performance of the inference server on a local machine, comparing its efficiency with and without GPU. This benchmark also allows us to estimate the hardware requirements for a

specified target load. To facilitate this, we employed the NVIDIA inference profiling tool, which allowed us to experiment with various inference server configuration parameters. This led to the creation of a Throughput/Latency trade-off graph. It is essential to note that this performance graph exclusively represents the cost of the inference and does not account for network or data loading overheads. Our results were obtained by sending parallel requests with random data to the inference server and then observing whether the server could sustain a consistent throughput at the designated rate.

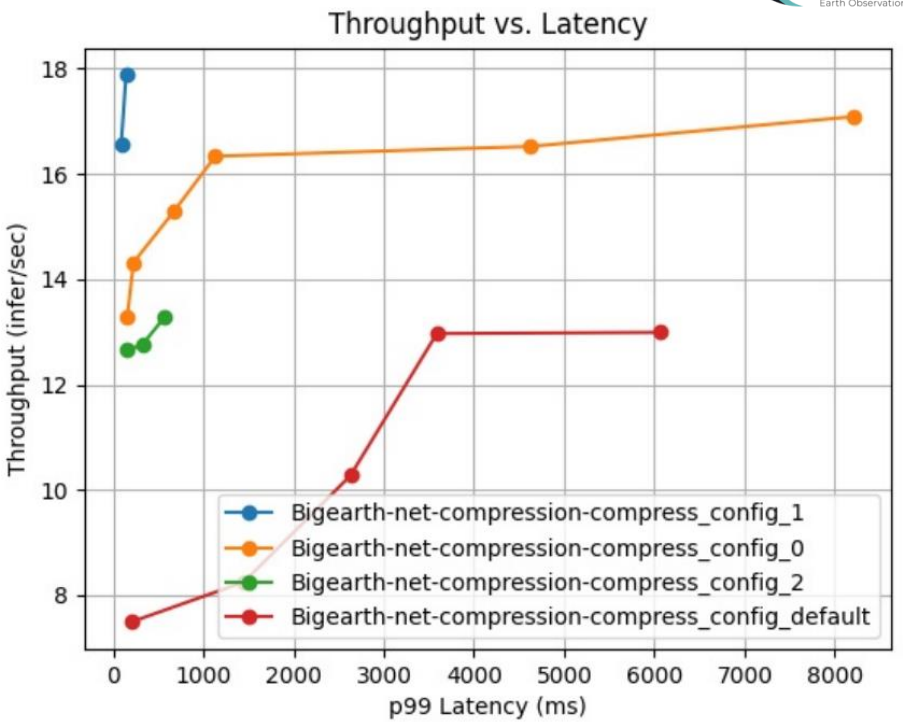
For the GPU-based experiments, we utilized 2 x GeForce 2080 TI. As seen in Figure 16, we observed a peak throughput of 300 inferences per second, accompanied by a latency just over 800ms.



Throughput vs. Latency curves for 3 best configurations.

Figure 16: Throughput vs. Latency with GPU.

For the CPU experiment, we utilized a workstation CPU with 10 physical cores: the Intel Core i9-9820X. As seen in Figure 17, we achieved a maximum throughput of 18 inferences per second and observed a latency of 143ms.



Throughput vs. Latency curves for 3 best configurations.

Figure 17:Throughput vs. Latency without GPU.

As observed in the figures, utilizing a GPU boosts the performance of the inference server by an order of magnitude. There is a noticeable throughput/latency trade-off. Nevertheless, the latency values we encountered are manageable for most applications, with the most significant delay being only 8 seconds.

Note that our infrastructure is horizontally scalable, with linear scaling. Doubling the number of nodes will double the throughput without significantly increasing the latency.

Given these findings, we recommend leveraging GPUs when dealing with substantial loads and resorting to CPUs for less demanding tasks or periods of lighter load.

4 Demonstration of model usage

In the previous part of the deliverable, we report on the development, evaluation, deployment of the task agnostic SSL and LC models for Sentinel-2 data, as well as the infrastructure that allowed for the integration and deployment of these models within the EO4EU infrastructure. As already discussed, the choice of Sentinel-2 stemmed from the fact that this is one of the most often used EO data while on the same time the availability of clearly defined supervised learning tasks together with the necessary annotations and labels allowed us to undertake the extensive evaluation of the reduction in annotation effort that one expect to attain.

Sentinel-2 data are used in UC5 (soil erosion) where the consortium has already developed and deployed baseline models in the EO4EU infrastructure. In addition to the baseline models, which do not use SSL features, we will train new downstream UC5 models that will make use of the available SSL-Sentinel-2 models and the features they extract. Finally, and importantly, the fact that the SSL-

Sentinel-2 models are already publicly available in EO4EU opens them up to the wider community, which can use them directly in their own supervised learning problems and tasks defined over Sentinel-2 data, allowing them to profit from the reduction of annotation effort they bring.

Going beyond Sentinel-2 data, we will also train, and make publicly available, SSL models at least on Copernicus ERA5 Land data, use them in the following versions of the use case 6 models (locust prediction), and compare them against the baseline UC6 models which are already available in the EO4EU platform. We should note here that UC6 models relied on extensive feature engineering and extraction; in that setting we also expect to benefit from the use of SSL features in terms of reductions in feature engineering efforts. In this perspective, the value added by SSL models to the platform goes far beyond the genericity and label-efficiency they were introduced for.

On the side of the learnt compression models, we will train models for additional EO data type such as ERA5. Model wise ERA5 data structure is very similar to Sentinel 2 so we expect no changes in the model architecture, with the probable exceptions of hyperparameter fine tuning. Moving to data with a 3D structure, e.g. with an additional dimension like altitude, we will need to move from 2D convolutions to 3D convolutions, a change that has the potential to introduce challenges with respect to the memory size of the models. More significant changes in the compression model architectures will be needed if the data structure is not a (2D or 3D or nD) grid, since one will not be able to use convolutions in such a case.

The use of learned compression in the platform will be transparent to the users of the platform and will serve within platform data transfers. As we make publicly available the trained compression models, we see an important opportunity for data providers using them, to compress data directly at the source, allowing for significant bandwidth reductions.

The consortium continues working on the development and training of ML models on the remaining use cases that have such needs. All the developed models will be deployed in the inference server, coupled with the data preprocessing and preparation workflows that feed the input data to the inference server. While the models are developed and described to be used by practitioners that have some experience in either EO data or ML, the consistent deployment of the solutions we implement on the inference server will also allow for the less experienced user of the platform to use the developed solutions. The different users will be supported through interactive notebooks (Jupyter, iPython) in which we lay out how to use the inference functionalities, e.g. to use SSL and/or the predictive models on their own data, whether within the EO4EU platform or by direct calls to the inference server, how to upload on the inference server their own models and use the data preprocessing pipelines to feed them data upon which inference will take place.

Below we comment briefly on the ML models already developed in UC5 and UC6, more detailed information on the use case specific models and results will be given in the different deliverables of WP5 that will document the work performed specifically on the use cases (D5.4, D5.5, D5.6).

Use case 5, soil erosion

Use Case 5 (UC5) operates directly with Sentinel-2 inputs to construct a regressor that targets an empirical measure of soil erosion, the "C-factor"; the C-factor assesses soil erosion on a scale of 0 to 1. Consequently, the machine learning models engineered to tackle UC5 are essentially predictors of the C-factor. As a part of the training data CMCC built a lookup table mapping spatial coordinates to C-factor value, using the Lukas dataset. For each of these coordinates we subsequently retrieved the

Sentinel-2 tile that contains them and cropped a smaller region centred on them. This allowed us to assemble a supervised dataset associating a C-factor for each of these tile crops. The dataset resulting from the pre-processing stage describe above contains 300,000 supervised pairs, and by design directly reflects the number of entries in the lookup stable assembled by UC5 partner, CMCC. The model architecture we used to learn the mapping from crop to scalar value is a 3-layer convolutional network followed by a 1-layer multi-layer perceptron. A depiction of the workflow described above is conveyed in Figure 18. We trained the model with the Adam optimizer, without learning rate scheduling.

The trained regressor has been deployed in the inference server and an inference pre-processing routine has been implemented and is available via the Fusion engine. In essence, from the DSL interface, one can run an inference job of the type we have just described by 1) selecting a Sentinel-2 data source via the Knowledge Graph, 2) select the "C-factor regressor" pre-processing routine in the Fusion engine, and finally 3) select the "C-factor regressor" ML task in the ML inference server.

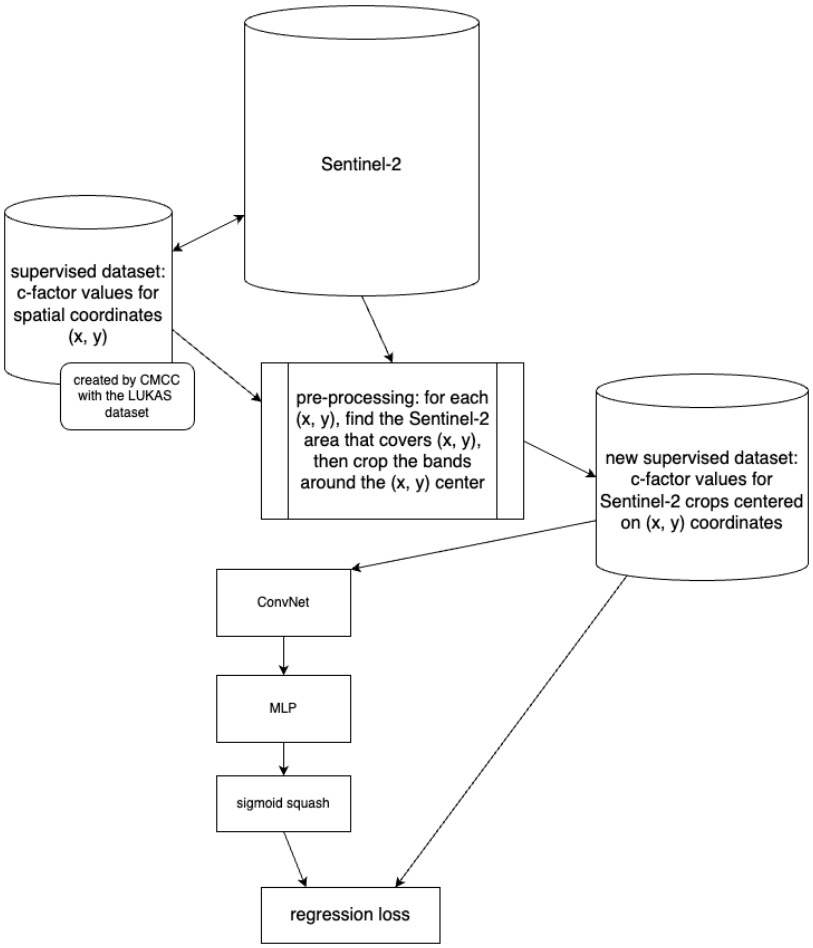


Figure 18: ML architecture of UC5

Use case 6, locust prediction

UC6 aims to deliver information services for assessing and predicting the impact of locust plagues leveraging EO and climate data, along with AI and machine learning techniques. The UC operates over a diverse set of data sources, including EO data (precipitation, soil moisture, vegetation tenure, land surface temperature), Copernicus ERA5 Land climate re-analyses, on-ground verification of locust

presence, and seasonal forecasts. The assembly of the input dataset that is to be used to predict locust breeding locations is a task carried out by UC6 partner, SISTEMA. Building the dataset involves various data sources across different dimensions, as well as the incorporation of a temporal dimension (the sources are laid out earlier in the paragraph). The consolidation work carried out by SISTEMA consisted in building temporal sequences of GeoTIFF tensors corresponding to rasters of statistics of precipitation and temperature values. In short, these describe the temperature and precipitation over a geographic grid. The dataset resulting from the described assembly consists of a set of temporal sequences of such GeoTIFF tensors, and each of these sequences traces how the tracked EO data metrics (precipitation and temperature) evolves over time. Coupled to these entries, the dataset can associate a flag designating the presence of locust breeding at the designated location. Notably, the locust breeding data sources only report the presence of locust and does not emit certainty about the absence of locust breeding when the location is not flagged. As such the task tackled in this use case is not a binary classification task (presence or absence of locust breeding) but a one-class classification task (akin to positive-unlabelled classification). The assembled dataset has 149,040 entries, with 1,799 positive samples for which locust breeding is flagged.OBJThe one-class classifier is modelled with [36]OBJ (in essence, a combination of convolutional networks or ConvNet and Long-Sort Term Memory networks or LSTM) and works over the inputFigure 19OBJ OBJLSTMs are a natural choice here as their ability to keep a recurrent inner summary makes them excel at extracting salient features from temporal series, such as the multi-dimensional raster sequences assembled here. By this choice of architecture, we hope that the model effectively captures the temporal trends underlying the input data. In practice, to deal with the absence of negative data (we only have information about the presence of locust breeding, the positive data), we assign negative labels to samples picked uniformly at random, while positive labels are only assigned to the samples that are known positive. This results in a conservative model that will tend to push the predicted labels to zero (the negative label, absence of locust) uniformly over the entire input space, while pushing the predicting labels to one (presence of locust) only on our positive samples. In effect, this is an approach sometimes referred to as pessimistic as it pushes the predictions to the predictions to the negative everywhere and pushes up when it is certain the label is positive. The pre-processing pipeline leading from source to the model inference execution is, as with UC5, fully available through the EO4EU platform.

[36] Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, *28*

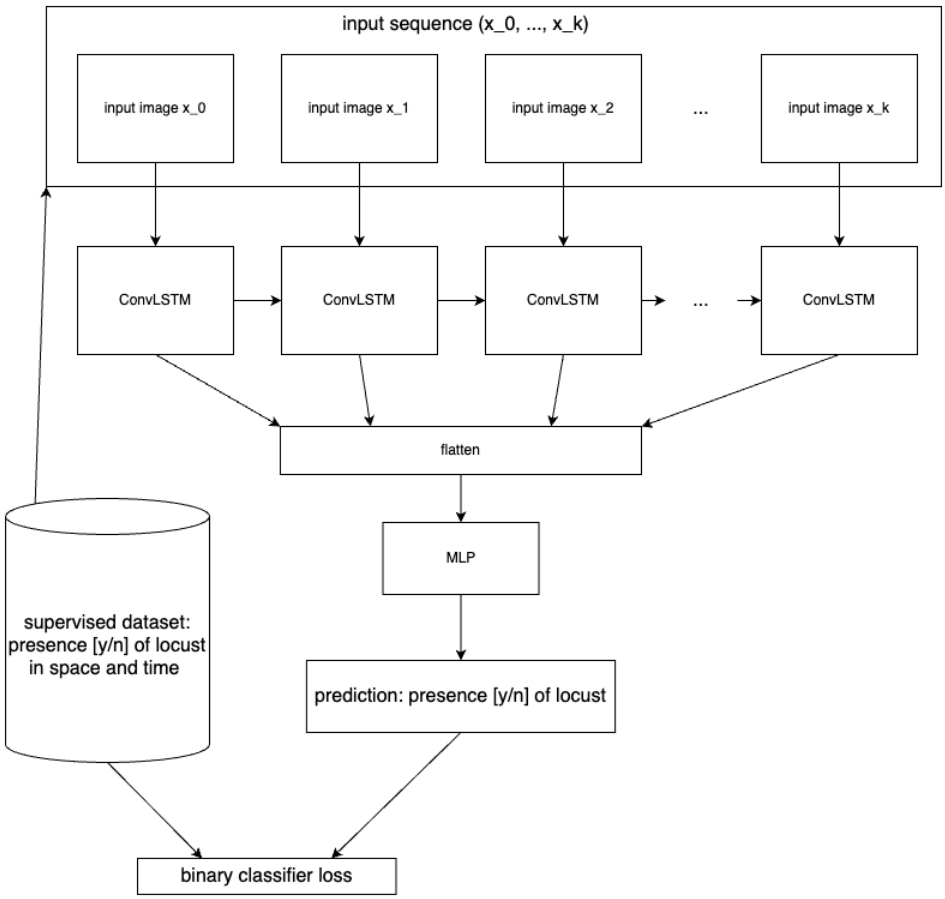


Figure 19: ML architecture of UC6

5 Impact of ML Models beyond EO4EU

The Machine Learning module of the EO4EU project, and its models described in this document, constitute one of several relevant project's modules that aim to facilitate the efficient accessing and processing of EO data by wider audiences. Throughout the project's progression, these models are showcased either by the comparison of their performance against established, well-known benchmarks or by the demonstration of their effectiveness as they assist the work of specific Use Cases within EO4EU. This strategic approach aims at ensuring that the value of these models resonates widely, benefiting users and other stakeholders within the EO domain, beyond the timeline of the project.

There are different ways that we envisage that the current work will help the interested user to profit from it. More specifically, users can profit from the already trained models that are exposed in EO4EU's platform and can also opt to utilise the code that was used to train these models, either for retraining (or for continuing the training of) the models with more data or by adapting the models' parameters according to their requirements. Independently of which of the previous options a user selects, the end goal is for these models to be used in a multitude of practical applications, facilitating users to overcome barriers related to EO data.

We will briefly elaborate on the way the available models can be used: a) a user uses one or several of the available models directly through the DSL, and b) a user chooses to build on one or several of our models from their open source code or from the model weights directly (further training the shared model according to possibly different objectives, in a downstream task fashion). User of type "a)" need not be an expert in neither EO nor ML. They can simply use the knowledge graph, marketplace, and DSL to build one's workflow, establishing a codeless pipeline between an EO data source, a preprocessing stage, and an ML model to run an inference task with. The provided ML documentation and code usage example can help this non-expert use steer through the available models to get a sense of which would assist one towards achieving one's ML inference objectives. At the same time the comprehensive model documentation, involving both training and inference examples, will also be instrumental for the design of ML-oriented workloads by an ML-expert user, who will, from a glance at the provided notebooks, know how to interface or build on the models with which the user might already be familiar with. Note, this user need not be an expert in EO data, as the notebooks show how one can pipe various types of inputs through the models. Both types of users will rely on the knowledge graph and GUI elements provided by the platform to get a sense of what the various types of EO data available through the knowledge graph represent. The user can then decide whether from their own expertise or via the involvement of a third party EO data expert, what pipeline to build and workflow to design to solve their task.

We here give an account of the two types of usage

A) Firstly, we consider the case in which users utilise directly the trained models, to facilitate their specific application. Regarding the SSL models and the significant reduction they induce in the annotation effort required for downstream tasks, there is an infinitely wide spectrum of applications one could consider. Indicatively, the most prominent domains of application that could be considered are nuanced tasks within environmental monitoring and disaster response, such as in agriculture and precision farming, climate change monitoring, coastal zone management, biodiversity and conservation, water resource management and others. Enabling a startup, a research group, and any other kind of individual or institutional stakeholder to be able to create a first prototype of their application, with a reduced volume of required labelled data for their application, could make a difference in their ability to continue their efforts and to capitalise on their work. For instance, a potential start-up proposing an application on crop quality detection, would be facilitated to reach a first prototype with a reduced volume of labelled data, using the SSL model trained on Sentinel-2 images. Moreover, along the evolution of the maturity of their application, the features provided by the SSL will remain a valuable resource assisting the further advancement of their modelling approach, be it with or without additional labelled data. This is a type of a straightforward scenario of direct usage of our models.
B) Secondly, the models could be evolved by the users. One way of doing so is by loading the model and continuing its training using further data that are available or data that concerns exclusively the use case of the user. For instance, the end user might have an application that concerns exclusively a specific type of land (mountainous regions, agricultural lands, coastlines), or a specific geographical region (a country, a continent). In this case, the SSL model could be fed with this specific type of EO data. It is possible for the training to continue, using the stored model. Moreover, the user can access the code of the model, and start the training from scratch, potentially adapting some parameters that they consider relevant. The same holds for the Learned Compression models, that the users have the freedom to train using the specific data modalities that interest them, so that they can profit from storage and bandwidth gains. Using more or fewer bands of satellite images, incorporating other modalities of EO data, or adapting the data augmentation routines (within SSL), are some

examples of adjustments that a knowledgeable user could do, to extend or adapt the capabilities of these models, in a way that is more tailored to their needs.

In addition to working with models that are already available in the platform, either reusing them as they are, or adapting them to one's own needs, one should not forget that the platform allows expert users to deploy their inference models and automate their data feeding from the data source to the inference server using the DSL and the functionalities it offers in orchestrating data pipelines (that involve data preprocessing, fusion, inference and visualisation). The inference as a service paradigm of EO4EU is one of the central contributions of the project and the platform.

6 Conclusion

In this deliverable, we presented the contributions of the Task "T3.3 - ML-based processing", in terms of produced ML models and methods. The emphasis of T3.3 was the development of task-agnostic models that bring significant reductions in annotation effort for downstream tasks and allow for information preserving compression of EO data. We reported on the development of the models and their evaluation in Section 2. We described how the developed models are integrated in the EO4EU platform in Section 3. In Section 4 we briefly touch upon some of the EO4EU Use Cases, the modelling that has been done on them, we defer the reader to the WP5 deliverables for more details on use-case specific models. Finally, in Section 5 we briefly discuss the significance and the potential use of machine learning models, within the context of EO4EU and even beyond it, highlighting potential user scenarios.

We clearly demonstrate significant reductions in annotation performance effort for downstream tasks using self-supervised models which are available in the platform. Such models can significantly reduce the cost of developing downstream task specific models. We also show significant compression rates using learned compression models which preserve to an important extent the original information content. Such compression models lead to significant reduction in bandwidth requirements when transferring data as well as in storage requirements.

Overall, the EO4EU platform makes possible the inference as a service paradigm for EO data. Users, depending on their level of expertise, can directly exploit the available models in their applications, they can adapt (retrain) the available models on their data and make them available in the EO4EU platform through the inference server, or develop their own models and deploy them through EO4EU inference server. The DSL language and the orchestration it allows of the different functionalities that the platform offers make very easy the feeding of data from the source to the inference server and towards downstream applications, significantly facilitating the deployment of EO centric applications.