remote sensing

MDPI

Article

AiTLAS: Artificial Intelligence Toolbox for Earth Observation

Ivica Dimitrovski [1,2], Ivan Kitanovski [1,2], Panče Panov [1,3], Ana Kostovska [1,3], Nikola Simidjievski [1,3,4] and Dragi Kocev [1,3,*]

[1] Bias Variance Labs, d.o.o., 1000 Ljubljana, Slovenia
[2] Faculty of Computer Science and Engineering, University Ss Cyril and Methodius, 1000 Skopje, North Macedonia
[3] Department of Knowledge Technologies, Jožef Stefan Institute, 1000 Ljubljana, Slovenia
[4] Department of Computer Science and Technology, University of Cambridge, Cambridge CB3 0FD, UK
* Correspondence: dragi@bvlabs.ai

Abstract: We propose AiTLAS—an open-source, state-of-the-art toolbox for exploratory and predictive analysis of satellite imagery. It implements a range of deep-learning architectures and models tailored for the EO tasks illustrated in this case. The versatility and applicability of the toolbox are showcased in a variety of EO tasks, including image scene classification, semantic image segmentation, object detection, and crop type prediction. These use cases demonstrate the potential of the toolbox to support the complete data analysis pipeline starting from data preparation and understanding, through learning novel models or fine-tuning existing ones, using models for making predictions on unseen images, and up to analysis and understanding of the predictions and the predictive performance yielded by the models. AiTLAS brings the AI and EO communities together by facilitating the use of EO data in the AI community and accelerating the uptake of (advanced) machine-learning methods and approaches by EO experts. It achieves this by providing: (1) user-friendly, accessible, and interoperable resources for data analysis through easily configurable and readily usable pipelines; (2) standardized, verifiable, and reusable data handling, wrangling, and pre-processing approaches for constructing AI-ready data; (3) modular and configurable modeling approaches and (pre-trained) models; and (4) standardized and reproducible benchmark protocols including data and models.

Keywords: Earth observation; remote sensing; deep learning; semantic segmentation; object detection; land use and land cover classification

Citation: Dimitrovski, I.; Kitanovski, I.; Panov, P.; Kostovska, A.; Simidjievski, N.; Kocev, D. AiTLAS: Artificial Intelligence Toolbox for Earth Observation. *Remote Sens.* **2023**, *15*, 2343. https://doi.org/10.3390/rs15092343

Academic Editor: Gwanggil Jeon

Received: 8 March 2023
Revised: 15 April 2023
Accepted: 24 April 2023
Published: 28 April 2023

1. Introduction

Remotely gathered data are available from a wide range of sources using a wide range of data collection techniques. Satellites, airplanes, and Unmanned Aerial Vehicles (UAVs) are equipped with various sensors that gather huge amounts of remotely sensed images that provide comprehensive spatial and temporal coverage of the Earth [1,2]. On the other hand, the increase in data production is well matched by the rapidly growing development of Artificial Intelligence (AI), which probes various aspects of natural sciences, technology, and society. Recent trends in machine learning, and particularly in deep learning, have ushered a new era of image analysis and raised the predictive performance bar in many application domains, including remote sensing and Earth observation [3]. Remote sensing data have been used in various application areas, including land use and land cover analysis [4], forest mapping [5,6], monitoring of natural hazards and disasters [7,8], precision agriculture [9], assessing the weather and observing climate changes [10], and various environmental studies [11].

With the ever-growing availability of remote sensing data, there has been a significant research effort to prepare, label, and provide proper datasets that will support the development and evaluation of sophisticated machine-learning methods [12–15]. In the past years, several publicly available high-resolution remote sensing image datasets have been made

available to support the research in a variety of remote sensing tasks, such as scene classification [4], semantic and instance segmentation [16], object detection [17], change detection, etc. Most of the annotated EO datasets are limited in scale and restricted in spatial coverage with a task-specific class distribution. Handling and pre-processing remote sensing image data can be very challenging due to their properties and heterogeneity, which greatly differ from conventional image data typically used in recent machine-learning pipelines. Namely, each type of sensor used for remote sensing has its own advantages (and disadvantages) conditioned by the geographical coverage, sensor resolution (spatial and temporal), and flight operations and specifics. For example, satellites are used for sensing at a global scale, and UAVs are typically used for sensing in small areas due to their flexibility and ease of operations in such conditions. Instead of 3-channel RGB imagery, the data in remote sensing are represented through different spectral, spatial, radiometric, and temporal resolutions:

- *Spectral resolution* defines the bandwidth and the sampling rate used to capture data. A high value for the spectral resolution means more narrow bands pertaining to small parts of the spectrum, and conversely, a low value means broader bands related to large parts of the spectrum. Spectral bands are groups of wavelengths, such as ultraviolet, visible, near-infrared, infrared, and microwave. Based on these, image sensors can be multi-spectral if they are able to cover tens of bands (e.g., Sentinel-2, which collects 12 bands) and hyper-spectral if they can collect thousands, such as Hyperion (part of the EO-1 satellite), which covers 220 spectral bands (0.4–2.5 μm) [18].
- *Spatial resolution* defines the size of the area on the Earth's surface represented by each pixel from an image. Spatial resolution relates to the level of detail captured in the image, with high resolutions (small pixel size) capturing more and low resolutions (large pixel size) capturing fewer details in an image. For example, most bands observed by the Moderate Resolution Imaging Spectroradiometer (MODIS) have a spatial resolution of 1 km, where each pixel represents a 1 km $\times$ 1 km area on the ground [19]. In contrast, images captured from UAVs or drones can have a very small spatial resolution of less than 1 cm [20].
- *Radiometric resolution* defines the number of discrete signals of given strengths that the sensor can record (also known as dynamic range). A large value of the dynamic range means that more details can be discerned in the recording, e.g., Landsat 7 records 8-bit images and can thus detect 256 unique gray values of the reflected energy [21]; similarly, Sentinel-2 has a 12-bit radiometric resolution (4095 gray values) [22]. In other words, a higher radiometric resolution allows for simultaneous observation of high and low-contrast objects in the scene. For example, a radiometric resolution is necessary to distinguish between subtle differences in ocean color when assessing water quality.
- *Temporal resolution* defines the frequency at which a given satellite revisits a given observation area. Polar-orbiting satellites have a temporal resolution that can vary from 1 day to 16 days (e.g., for Sentinel-2, this is ten days [22]). The temporal aspects of remote sensing are essential in monitoring and detecting changes in given observation areas (incl. land use change, mowing, and deforestation).

The increasing amount of available EO data is equally matched with the number of libraries and toolboxes designed to handle, process, and potentially provide a data analysis framework for such data. However, considering the complexity of EO data coupled with the diversity of EO tasks that can be addressed, many of the available libraries and toolboxes focus either on the data-specific processing aspect, have a very narrow application horizon, or include machine-learning approaches that are hardly accessible for domain experts. This relates to libraries such as eo-learn (open source) [23] and Up42 (commercial product) [24], which provide accessible means for EO data processing/feature extraction workflows from satellite imagery. These libraries focus mainly on the data acquisition, handling, and data pre-processing stages of the workflow and offer limited machine-learning capabilities. Similarly, Sentinels for Common Agriculture Policy (Sen4CAP), an open-source project based on the Sentinel Application Platform (SNAP), provides data

pre-processing algorithms and workflows but only in the limited scope of agriculture monitoring relevant to the management of the CAP [25].

More recent libraries such as Orfeo ToolBox (OTB) [26] offer similar capabilities with respect to data pre-processing, data augmentation, and feature extraction pipelines, in addition to a catalog of more traditional machine-learning approaches for image data analysis. OTB further allows remote integration with deep-learning libraries such as TensorFlow [27]. However, this capability is part of an unofficial OTB module aimed primarily at AI users. On the other hand, CANDELA [28] is an end-to-end platform tailored for EO users, focusing on services that provide quick data access and exploratory data analysis. In addition to the core data handling capabilities, CANDELA allows for handcrafted application-centered data analysis blocks that employ data pre-processing and machine-learning methods but are specifically tailored for a particular EO task at hand (such as change detection). TorchGeo [29] is a recent library that builds on the PyTorch [30] deep-learning framework that includes more general methods for EO data analysis. Namely, it includes data loaders for standard benchmark datasets, methods for data handling, and data transformations, as well as a catalog of (pre-trained) vision models applicable to different tasks pertaining to EO applications.

An implicit but common theme among most of these libraries is the community barrier. Libraries that offer the most recent machine-learning approaches are tailored for data scientists and have a steep learning curve for domain experts, but libraries tailored for the remote sensing community are not easily applicable in modern machine-learning pipelines. While most of these attempts are a step in the right direction, there is still a gap related to the need for a common AI4EO framework that will provide (1) accessible and interoperability resources for data analysis (via configurable and readily usable pipelines); (2) standardized, verifiable, and reusable data handling, wrangling, and pre-processing approaches for constructing AI-ready data; (3) modular and configurable modeling approaches and (pre-trained) models; and (4) standardized and reproducible benchmark protocols (including data and models).

We present AiTLAS (http://aitlas.bvlabs.ai (accessed on 8 March 2023)), an open-source AI4EO toolbox that is designed based on the principles outlined above, thus facilitating the use of EO data in the AI community and, more importantly, accelerating the uptake of (advanced) machine-learning methods and approaches by EO experts. AiTLAS provides various resources, including customizable and easily usable data analysis pipelines; semantically annotated datasets, formalized to be used directly by AI methods; recent approaches for learning models de novo coupled with a model catalog consisting of large pre-trained vision models applicable to EO tasks; standardized frameworks for model benchmarking [3]; mechanisms for quantitative and qualitative model evaluation, etc.

In this paper, we provide extensive details of the many functionalities and capabilities of AiTLAS. We demonstrate its versatility for use in different EO tasks by exploiting the variety of EO data and AI methods made available for direct use within the toolbox. We first explain the design and implementation of AiTLAS and then discuss the supported EO tasks and data. Next, we explain the AI methods that are implemented within the toolbox. Furthermore, we showcase several use cases to illustrate the basic principles behind the toolbox, its modularity, and its flexibility. Lastly, we summarize the distinctive proprieties of AiTLAS and outline directions for its further development.

2. Materials and Methods

2.1. Design and Implementation of the AiTLAS Toolbox

The AiTLAS toolbox is designed such that leveraging recent (and sophisticated) deep-learning approaches over a variety of EO tasks (and data) is straightforward. On the one hand, it utilizes EO data resources in an AI-ready form; on the other hand, it provides a sufficient layer of abstraction for building and executing data analysis pipelines, thus facilitating better usability and accessibility of the underlying approaches—particularly useful for users with limited experience in machine learning, and in particular deep learning.

AiTLAS can be used both as an end-to-end standalone tool and as a *modular* library. Users can use and build on different toolbox components independently, be they related to the tasks, datasets, models, benchmarks, or complete pipelines. It is also *flexible* and *versatile*, facilitating the execution of a wide array of tasks on various domains and providing easy extension and adaptation to novel tasks and domains. Moreover, AiTLAS adheres to the principle *less is more*—it embeds the most common tasks and functionalities in easy-to-use interfaces that simplify the usage and adaptation of the toolbox with minimal modifications. Last but not least, AiTLAS is fully aligned with the principles of *open science*—its development is community-driven and open-source.

Figure 1 presents a high-level schematic diagram of the main modules and components of AiTLAS. It is designed around the concept of a workflow, where users need to define a specific task (`aitlas.tasks`), be it an exploratory analysis of a dataset or a predictive task of a different kind, such as image classification, object detection, image segmentation, etc. In turn, the instantiated task serves as an arbiter of the workflow and orchestrates the flow between the two central components of the toolbox—the datasets (`aitlas.datasets`) and the models (`aitlas.models`)—which relate to AI-ready formalized data and configurable model architectures, respectively. Programmatically, these modules are embedded within the core module `aitlas.base`, which contains all main abstract definitions related to every module, such as definitions of tasks, models, and datasets, but are also related to evaluations (`aitlas.metrics`), data transformations (`aitlas.transforms`), and various types of visualizations (`aitlas.visulizataions` and `aitlas.datasets.visulizataions`).

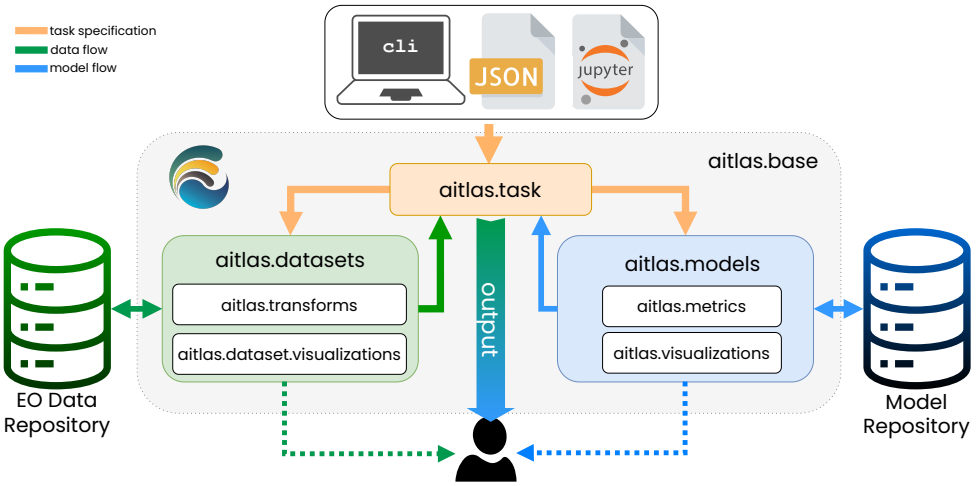


Figure 1. Diagram of the main modules and components in the AiTLAS toolbox.

More specifically, as a standalone application, the flow of AiTLAS begins with the user-specified definition of a task. These definitions can be provided at input via *command-line interface* (`CLI`) as a formatted `JSON` configuration file or more directly executed via Jupiter notebooks. This initiates the arbiter module `aitlas.tasks`. The `aitlas.tasks` act as a controller and component mediator during the entire workflow. To this end, AiTLAS can handle a variety of typical workflows, such as training and evaluating a model, data pre-processing and calculating statistics, extracting features, etc.; while the implemented tasks can be applied in many different scenarios, they can also serve as a blueprint for creating and instantiating new, more specific, tasks.

Typically, a *task* instantiates a specific *dataset* component as per the configuration and prepares it for processing. The *dataset* components, implemented in the `aitlas.datasets` module, encapsulate different operations for working with the underlying EO data, such as reading and writing from and to storage and preparing it for further processing. Each EO dataset has a separate and specific implementation of its *dataset* component since the different datasets have different formats, organizational structures, etc. (Tables 1–5 provide a list of currently supported datasets). Note that the datasets must be accessible for the machine that executes AiTLAS. Therefore, it is up to the user to download the datasets they work with, and organize their access, as AiTLAS only provides the means to access a dataset.

Once the data access is ready, AiTLAS offers various mechanisms for pre-processing, handling, and transforming raw EO data into an AI-ready format. These mechanisms are implemented in the `aitlas.transforms` module. More specifically, besides standard functions for handling image data, this module contains specific implementations of augmentations and transformations that can be applied to images, such as rotations, resizing, cropping, etc. These transformations can be configured to be applied to any raw image (regardless of a task), including target masks, as in the case of image segmentation. The processed (AI-ready) data is, in turn, used in the workflow. Moreover, for better reusability and reproducibility, AiTLAS also annotates and can store the processed data (with the accompanied meta-data) in an EO data repository such as the *AiTLAS semantic data catalog* (http://eodata.bvlabs.ai (accessed on 8 March 2023)), where users can further analyze and query the available data.

The *task* component also interacts with the *model* components within the `aitlas.models` module. The models in AiTLAS are based on the PyTorch framework [30]. The *model* component wraps the architecture of the deep-learning model and only exposes the operations for controlling their behavior, such as training the model, using it to perform predictions, saving it, or loading it from storage, etc. The `aitlas.models` module contains concrete implementations of the deep-learning models providing the means to work with individual model architectures (see Table 6 for a current list of implemented architectures). The concrete implementations are responsible for model instantiation and forwarding the input data. In the case of pre-trained models, the specific implementation can pull a remote or local version of the pre-trained model.

The models' performance is estimated using the *metrics* components implemented in the `aitlas.metrics` module. Depending on the task at hand (classification, segmentation), the module offers a variety of evaluation measures to assess the performance of the models, such as accuracy, F1 score, etc. Note that similar to storing processed data, AiTLAS also supports storing trained models. To this end, the AiTLAS model catalog [3] contains more than 500 trained models for EO image scene classification, trained and evaluated on 22 different EO datasets.

AiTLAS allows for certain aspects of the workflow to be illustrated, fostering better user interaction with the learning process and more interpretable outcomes. This is enabled via the *visualizations* components implemented in the `aitlas.visualization` and `aitlas.dataset.visualization` modules. These components provide the means to provide further inspection and analysis via visualizing datasets (samples and properties), tracking model performance, and visualizing model predictions.

AiTLAS is built in Python and uses a variety of other libraries related to different parts of the project. The core underlying library is PyTorch [30]—the AiTLAS model architecture extends PyTorch's model class. The extensions add the means for training, evaluating, predicting, resource utilization, saving, and loading the model from disk. The AiTLAS dataset management also extends the data module from PyTorch. As previously stated, AiTLAS can be used both as a standalone application and as a library embedded within other projects. The remaining dependencies are given in Table A1 in Appendix A, with further details of their scope of usage.

2.2. EO Data and Common Tasks

The AiTLAS toolbox can be applied for a variety of EO tasks (and datasets). For clarity, we present and discuss four common types of workflows pertaining to typical EO tasks: (1) image scene classification, (2) image semantic segmentation, (3) image object detection, and (4) crop type prediction using satellite time series data.

2.2.1. Image Scene Classification Tasks

The task of image scene classification refers to annotating images. In a typical scenario, working with large-scale EO images, this task addresses classifying smaller images (patches) extracted from a much larger remote sensing image. The extracted images can then be annotated based on the content using explicit semantic classes (e.g. forests, residential areas, rivers, etc.). Given an image as an input, the output would be single or multiple annotations with semantic labels, denoting land-use and/or land-cover (LULC) classes present in that image, as illustrated in Figure 2.

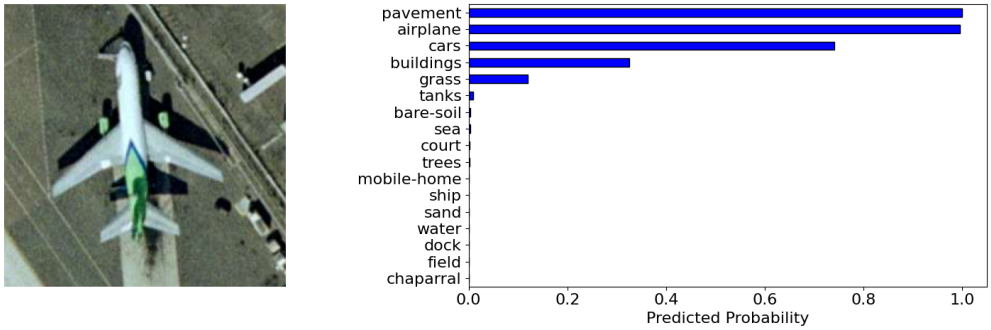


Figure 2. Remote sensing image scene classification: sample image patch provided on the left and the output (predicted LULC classes) shown on the right subfigure. The image is a sample from the UC Merced dataset from the MLC task [31].

Based on the number of semantic labels assigned to the images in the datasets, image scene classification tasks can be further divided into multi-class (MCC) and multi-label (MLC) classification. In the *multi-class classification* setting, each image is associated with a single class (label) from a set of predefined classes. The goal, in this case, is predicting one (and only one) class for each image in the dataset. In the *multi-label classification* setting, on the other hand, images are associated with multiple labels (from a predefined set) based on the information. The goal is then to predict the complete set of labels for each image in the dataset at hand [32]. To this end, AiTLAS offers 22 such datasets that can be readily used for a variety of MLC and MCC modeling tasks. These also serve as a blueprint for applying AiTLAS to other datasets pertaining to similar tasks. Tables 1 and 2 summarize the properties of the considered MCC and MLC datasets, respectively. The number of images across datasets can be quite diverse, ranging from datasets with ∼2 K images to datasets with ∼500 K images. This also holds for the number of labels per image, ranging from 2 to 60. Most of the datasets are comprised of aerial RGB images (with only a few comprised of satellite multi-spectral data) that are different in spatial resolution, size, and format. Finally, note that AiTLAS also provides standardized procedures for analyzing these datasets in terms of predefined splits (for training, validation, and testing), which will ensure reusable and reproducible experiments. A more detailed description of each dataset can be found in [3].

Table 1. Properties of the multi-class image scene classification (MCC) datasets available in the AiTLAS toolbox.

Name	Image Type	#Images	Image Size	Spatial Resolution	#Labels	Predefined Splits	Image Format
UC Merced [33]	Aerial RGB	2100	256 × 256	0.3 m	21	No	tif
WHU-RS19 [34]	Aerial RGB	1005	600 × 600	0.5 m	19	No	jpg
AID [35]	Aerial RGB	10,000	600 × 600	0.5 m–8 m	30	No	jpg
Eurosat [36]	Sat. Multispectral	27,000	64 × 64	10 m	10	No	jpg/tif
PatternNet [37]	Aerial RGB	30,400	256 × 256	0.06 m–4.69 m	38	No	jpg
Resisc45 [4]	Aerial RGB	31,500	256 × 256	0.2 m–30 m	45	No	jpg
RSI-CB256 [38]	Aerial RGB	24,747	256 × 256	0.3–3 m	35	No	tif
RSSCN7 [39]	Aerial RGB	2800	400 × 400	n/a	7	No	jpg
SAT6 [40]	RGB + NIR	405,000	28 × 28	1 m	6	Yes	mat
Siri-Whu [41]	Aerial RGB	2400	200 × 200	2 m	12	No	tif
CLRS [42]	Aerial RGB	15,000	256 × 256	0.26 m–8.85 m	25	No	tif
RSD46-WHU [43]	Aerial RGB	116,893	256 × 256	0.5 m–2 m	46	Yes	jpg
Optimal 31 [44]	Aerial RGB	1860	256 × 256	n/a	31	No	jpg
Brazilian Coffee Scenes (BSC) [45]	Aerial RGB	2876	64 × 64	10 m	2	No	jpg
SO2Sat [46]	Sat. Multispectral	400,673	32 × 32	10 m	17	Yes	h5

Table 2. Properties of the multi-label image scene classification (MLC) datasets available in the AiTLAS toolbox.

Name	Image Type	#Images	Image Size	Spatial Resolution	#Labels	#Labels per Image	Predefined Splits	Image Format
UC Merced (mlc) [31]	Aerial RGB	2100	256 × 256	0.3 m	17	3.3	No	tif
MLRSNet [47]	Aerial RGB	109,161	256 × 256	0.1 m–10 m	60	5.0	No	jpg
DFC15 [48]	Aerial RGB	3342	600 × 600 20 × 20 60 × 60	0.05 m 60 m 20 m	8	2.8	Yes	png
BigEarthNet 19 [13]	Sat. Multispectral	519,284	120 × 120 20 × 20 60 × 60	10 m 60 m 20 m	19	2.9	Yes	tif, json
BigEarthNet 43 [49]	Sat. Multispectral	519,284	120 × 120	10 m	43	3.0	Yes	tif, json
AID (mlc) [50]	Aerial RGB	3000	600 × 600	0.5 m–8 m	17	5.2	Yes	jpg
PlanetUAS [51]	Aerial RGB	40,479	256 × 256	3 m	17	2.9	No	jpg/tiff

2.2.2. Object Detection Tasks

Object detection is another common EO task that focuses on identifying and localizing objects present in an image. In the typical setting, this relates to annotating the identified objects with respect to different predefined classes and providing their location on the image (as a bounding box). Figure 3 illustrates an example of object detection, i.e., detecting ships at sea.

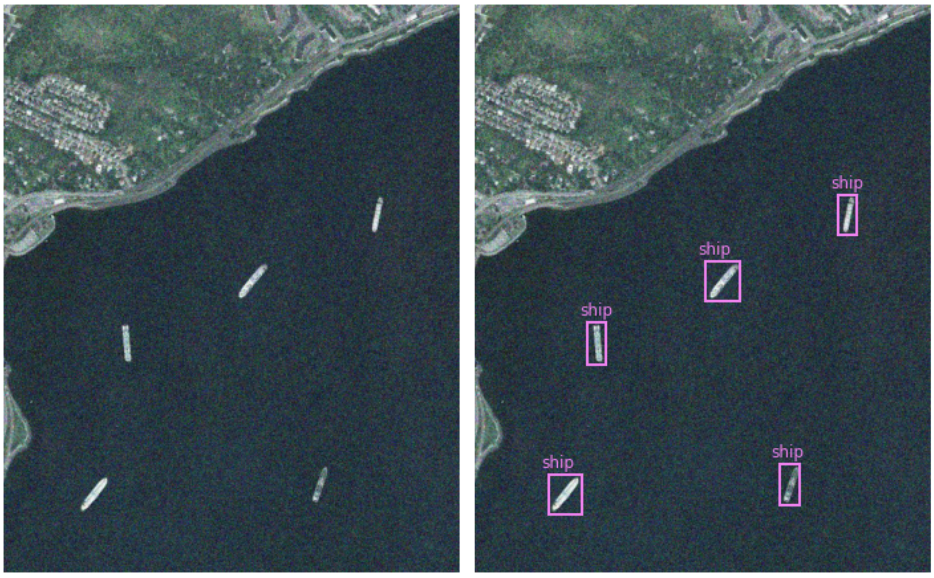


Figure 3. Remote sensing image object detection: sample image provided on the left and in the output image on the right, the objects are detected and localized with bounding boxes. The image is a sample from the HRRSD dataset [52].

Common instances of this task include the detection of specific objects such as buildings, vehicles, ships, and planes [53,54] from aerial image datasets. The AiTLAS toolbox readily supports such tasks and datasets, which follow the Pascal VOC and the COCO (Common Objects in Context) formatting guidelines for object annotations. To this end, it offers four such datasets (Table 3) for development and benchmarking object detection methods.

Table 3. Properties of the object detection datasets available in the AiTLAS toolbox.

Name	Image Type	#Images	#Instances	#Labels	Image Width	Spatial Resolution	Image Format
HRRSD [52]	Aerial RGB	21,761	55,740	13	152–10,569	0.15–1.2 m	jpeg
DIOR [54]	Aerial RGB	23,463	192,472	20	800	0.5–30 m	jpeg
NWPU VHR-10 [55]	Aerial RGB	800	3651	10	~800	0.08–2 m	jpeg
SIMD [56]	Aerial RGB	5000	45,096	15	1024	0.15–0.35 m	jpeg

2.2.3. Image Semantic Segmentation Tasks

The tasks of image semantic segmentation aim at the fine-grained identification of objects in an image. In contrast to object detection, which aims at coarser localization of the detected objects, segmentation tasks focus on labeling each pixel of an image with a corresponding class of what the pixel represents. In the typical scenario, the input is an image, and the output is a mask (overlay) of categorized pixels based on a single semantic type present in the image. Figure 4 illustrates an example of image semantic segmentation. The more sophisticated extension of semantic segmentation tasks, referred to as *instance segmentation*, takes into account different semantic types and focuses on delineating multiple objects present in an image.

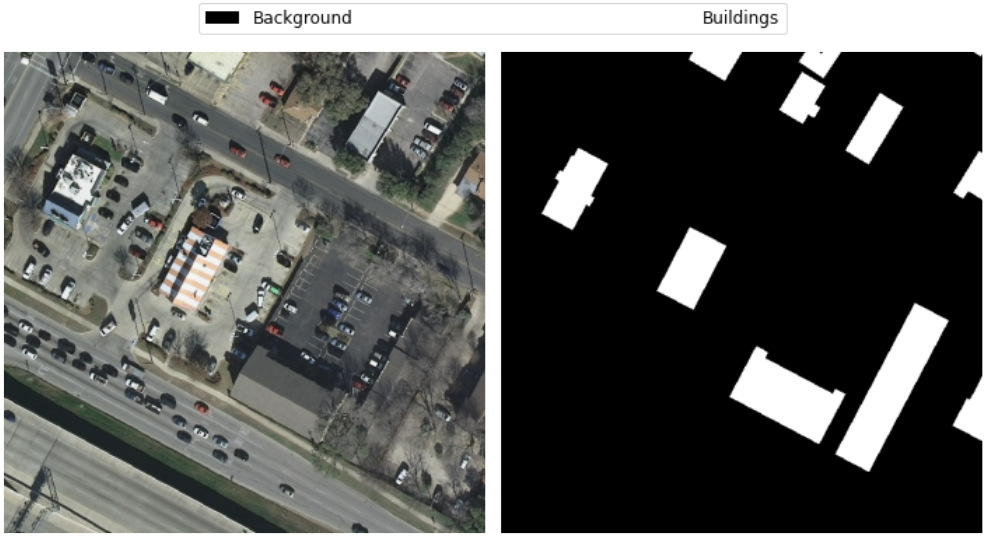


Figure 4. Remote sensing image semantic segmentation of buildings: sample image provided on the left and the output, which is the overlay mask of predictions on the right. The image is a sample from the Massachusetts Buildings dataset [57].

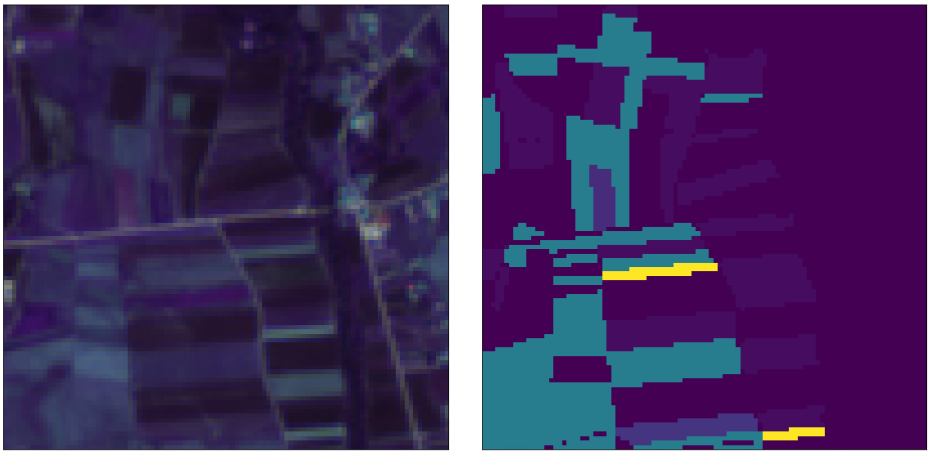
The AiTLAS toolbox offers several such datasets (summarized in Table 4) that can be readily used for EO image semantic segmentation tasks. All but one of the datasets are comprised of aerial RGB images (the remaining contains satellite multi-spectral data) with different spatial resolutions and sizes. The number of semantic labels in these datasets ranges from 2 to 5.

Table 4. Properties of the semantic segmentation datasets available in the AiTLAS toolbox.

Name	Image Type	#Images	Image Size	Spatial Resolution	#Labels	Image Format
LandCover.ai [58]	Aerial RGB	41	4200 × 4700 9000 × 9500	0.25–0.5 m	5	geo tif
Inria [59]	Aerial RGB	360	5000 × 5000	0.3 m	2	tif
AIRS [60]	Aerial RGB	1047	10,000 × 10,000	0.075 m	2	tif
Amazon Rainforest [61]	Aerial RGB	60	512 × 512	n/a	2	geo tif
Chactun [62]	Sat. Multispectral	2093	480 × 480	10 m	3	geo tiff
Massachusetts Roads [57]	Aerial RGB	1171	1500 × 1500	1 m	2	tiff
Massachusetts Buildings [57]	Aerial RGB	151	1500 × 1500	1 m	2	tiff

2.2.4. Crop Type Prediction Tasks

Crop type prediction is a semantic segmentation task that aims to map vegetation on the crops present in a given area. The main difference with the classical semantic segmentation task is that crop type prediction necessarily involves a temporal component. Namely, to properly train a model, it needs to be presented with data of the same area over different periods in time (preferably covering the whole growing season, e.g., the periods with longer daytime and more sunlight). The added complexity is that there is variability between different periods and locations (among different countries or even within the same country) as there is variability between different years. The key feature of any crop-type detection method is to utilize both the spatial and temporal data in multi-temporal satellite imagery. The input in this task is multi-temporal satellite imagery data for a specific geographic area. The output is a segmented mapping of the present crops in that geographic area—Figure 5 illustrates an example of this task.



(**a**) (**b**)

Figure 5. Crop type prediction: The provided patch is represented by the image (**a**), and the output is the overlay mask of predictions (**b**). The image is a sample from the AiTLAS NLD dataset [63].

Datasets for crop type prediction are generally spatio-temporal, i.e., they contain time series data in addition to the EO image data. Table 5 presents several datasets available within the AiTLAS toolbox. On top of the recent Breizhcrops dataset [64], AiTLAS also presents novel Sentinel 2 imagery for three European countries (Denmark, the Netherlands, and Slovenia) across three years—2017, 2018, and 2019. These novel datasets are significant due to their size w.r.t. the geographic area they cover, the number of different parcels (i.e., polygons), and the number of distinct crop fields. The datasets are presented in detail in [63].

Table 5. Properties of the crop type prediction datasets available in the AiTLAS toolbox.

Dataset	# of Polygons	Area Covered	# of Crop Types
AiTLAS SLO [63]	800 k	5000 km^2	27
AiTLAS DNK [63]	580 k	26,000 km^2	27
AiTLAS NLD [63]	750 k	18,000 km^2	27
Breizhcrops [64]	580 k	27,200 km^2	9

2.3. Model Architectures

The AiTLAS toolbox contains a catalog of deep learning (DL) model architectures that support different EO tasks, including image classification, semantic segmentation, object detection, and crop-type prediction. To this end, AiTLAS implements 24 model architectures, listed in Table 6. For each model, we present the basis of its implementation, technical characteristics, and supported tasks. In the remainder, we discuss the different available architectures in AiTLAS, the basis of their implementation and technical characteristics, and the EO tasks to which they are applicable.

Table 6. Deep neural network architectures implemented in AiTLAS and their usability across the EO tasks.

Model	Supported Tasks				Based on
	Im. Scene Class.	Seman. Segm.	Obj. Detection	Crop Type Pred.	
AlexNet [65]	✓				[66]
CNN-RNN [67]	✓				[67]
ConvNeXt [68]	✓				[66]
DenseNet161 [69]	✓				[66]
EfficientNet [70]	✓				[66]
MLPMixer [71]	✓				[72]
ResNet152 [73]	✓				[66]
ResNet50 [73]	✓				[66]
Swin Transformer [74]	✓				[66]
VGG16 [75]	✓				[66]
Vision Transformer [76]	✓				[72]
DeepLabV3 [77]		✓			[66]
DeepLabV3+ [78]		✓			[79]
FCN [80]		✓			[66]
HRNet [81]		✓			[72]
UNet [82]		✓			[79]
RetinaNet [83]			✓		[66]
Faster R-CNN [84]			✓		[66]
InceptionTime [85]				✓	[86]
LSTM [87]				✓	[86]
MSResNet [88]				✓	[86]
OmniScaleCNN [89]				✓	[86]
StarRNN [90]				✓	[86]
TempCNN [91]				✓	[86]
Transformer for time series classification [92]				✓	[86]

Regarding **image scene classification tasks**, AiTLAS implements a variety of well-known DL models based on the traditional convolutional architectures, but also the more recent attention-based and mlp-based architectures. *Convolutional DL architectures* have contributed to many advances in computer vision. A convolutional neural network (CNN) typically consists of many (hidden) layers stacked together, designed to process (image) data in the form of multiple arrays. The distinctive component in these networks is the convolutional layers, which apply the convolution operation (passing the data through a kernel/filter) and forward the output to the next layer. This serves as a mechanism for constructing feature maps, with former layers typically learning low-level features (such as edges and contours) and subsequently increasing the complexity of the learned features with deeper layers in the network.

The convolutional layers are typically followed by pooling operations (serving as a downsampling mechanism) that aggregate the feature maps through local non-linear operations. In turn, these feature maps are fed to fully-connected layers which perform the ML task at hand—in this case, classification. All the layers in a network employ an activation function. In practice, the intermediate hidden layers employ a non-linear function such as rectified linear unit (ReLU) or Gaussian Error Linear Unit (GELU) as common choices. The choice of activation function in the final layer relates to the tasks at hand—typically, this is a sigmoid function in the case of classification. CNN architectures can also include different normalization and/or dropout operators embedded among the different layers, which can further improve the network's performance. CNNs have been

extensively researched, with models applied in many contexts of remote sensing, and in particular EO image classification [93–96].

Recently, *attention-based network architectures* have shown state-of-the-art performance in various vision tasks, including tasks in EO domains. Very prominent in this aspect is the *Vision Transformers* (ViT) [76]—they are inspired by the popular NLP (natural language processing) transformer architecture [97], and leverage the attention mechanism for vision tasks. Much like the original transformer architecture that seeks to learn implicit relationships in sequences of word tokens via multi-head self-attention, ViTs focus on learning such relationships between image patches. Typically, ViTs employ a standard transformer encoder that takes a lower-dimensional (linear) representation of these image patches together with additional positional embedding from each, in turn feeding the encoder output to a standard MLP head. More recent and sophisticated attention network architectures such as the *Swin Transformers* (SwinT) [74,98] rely on additional visual inductive biases by introducing hierarchy, translation invariance, and locality in the attention mechanism. ViT and SwinT variants have shown excellent performance in practice on various vision tasks, including EO applications [3,99–101], particularly when pre-trained with large image datasets.

An attention mechanism can be obtained with different approaches, e.g., attending over channels and/or spatial information, and with convolutional architectures [102–104]. Another alternative is the *MLPMixer* [71]—it obtains its attention mechanism relying on the classical MLP architecture. Namely, similarly to a transformer architecture, an MLPMixer operates on image patches. It consists of two main components: a block of MLP layers for 'mixing' the spatial patch-level information on every channel and a block of MLP layers for 'mixing' the channel information of an image. This renders lightweight models, with performance on par with many much more sophisticated architectures [96,105,106].

The **semantic segmentation** tasks refer to pixel-wise classification [16]. In this scenario, segmentation models typically learn to extract meaningful features/representations from an image and use them to separate the image into multiple segments. In this context, convolutional architectures are frequently used for performing this task. As in the typical convolutional setting, the image is first passed through a series of layers (that learn image features). This process downsamples the image as it passes through a series of pooling layers. In turn, the image is upsampled/interpolated back to its original size (typically by using deconvolutional layers), but with some loss of information. The output is typically passed to a convolutional block with a sigmoid activation, which provides the resulting pixel-wise classification of the image. This relates to pixel-to-pixel fully convolutional networks (FCN) [80]. More sophisticated segmentation architectures typically combine existing architectures at different stages, such as for the model's downsampling (encoding), upsampling (decoding), and prediction blocks.

AiTLAS supports segmentation tasks and implements a variety of state-of-the-art architectures with a track record of successful applications for semantic image segmentation. This includes *UNet* [82], a robust, versatile, and accurate segmentation architecture [107]. UNet is a modification of FCN, consisting of encoder and decoder blocks (in a U shape): the encoder blocks relate the extracted features to the corresponding blocks of the decoder, with an additional shortcut connection in the decoder. This allows the model to capture more specific information from the image and retain more information by concatenating high-level features with low-level ones. AiTLAS also employs HRNet (High-Resolution Net), another fully convolutional network [81] with parallel architecture and multiple group convolutions. This allows for leveraging high-resolution images, which leads to better performance [108].

Common segmentation architectures employ a downsampling block, which broadens the receptive field (given the input) for the forthcoming filter(s), but at the cost of reduced spatial resolution. An alternative approach with the same effect but with the ability to preserve the spatial resolution is atrous (or dilated) convolutions. Here, the filter is upsampled along each spatial dimension by inserting zero values between two successive

filters. The *DeepLab* segmentation architectures [109] employ this approach, incorporating both atrous convolutions and atrous spatial pyramid pooling (ASPP), leading to robust and accurate performance semantic segmentation of high-resolution images. To this end, AiTLAS implements DeepLabv3 [77] and DeepLabv3+ [78] architectures that have been successfully applied in a variety of EO application [110,111].

Another class of common tasks in EO domains is **object detection**, which aims at the localization and classification of objects present in an image. Typical deep-learning approaches that address object detection tasks can be divided into two groups: region proposal-based and regression-based [54]. Region proposal-based approaches tackle object detection in two stages. The first stage focuses on generating a series of candidate region proposals that may contain objects. The second stage classifies the candidate region proposals obtained from the first stage into object classes or backgrounds and further fine-tunes the coordinates of the bounding boxes. In contrast, *regression-based approaches* transform the problem to a multi-target regression task, focusing on directly predicting the coordinates of the (detection) bounding box.

In the domain of EO, object detection approaches are typically applied on aerial images, which can be challenging due to the significant variation in scale and viewpoint across the diverse set of object categories [53]. Most studies involving aerial images use region proposal-based methods to detect multi-class objects. To support these tasks, AiTLAS implements several state-of-the-art architectures such as the improved Faster R-CNN model with a ResNet-50-FPN backbone [84,112] and RetinaNet with ResNet-50-FPN backbone [83].

Finally, AiTLAS also provides approaches for addressing tasks of **crop type prediction**. This refers to a multi-dimensional time series classification task where the input is multi-spectral temporal data and the output is a discrete variable specifying the crop type; while these tasks have traditionally been tackled using standard machine-learning approaches (such as Random Forest [113]), more recent deep-learning approaches have shown better results [64]. These include approaches that build on convolutional, recurrence, and self-attention-based architectures.

The convolution-based models use a one-dimensional convolutional layer to extract features from a temporal local neighborhood by convolving the input time series with a filter bank learned by gradient descent. To this end, AiTLAS provides implementations of several such architectures that have been successfully used for crop type prediction and land cover mapping [91,114], including Temporal Convolutional Neural Network (TempCNN) [91] (TempCNN), Multi-Scale 1D Residual Network (MSResNet) [88], InceptionTime [85], and Omniscale Convolutional Neural Network (OmniscaleCNN) [89]. Recurrent Neural Network (RNN) models process a series of observations sequentially while maintaining a feature representation from the previous context. AiTLAS provides implementations for Long Short-Term Memory (LSTM) [115] models, which have been successfully used in remote sensing applications, especially for land cover mapping [116,117]. Finally, AiTLAS includes recent state-of-the-art attention-based transformer architectures based on [92], which can learn and use the most relevant parts of the input sequence via stacked self-attention layers for sequence-to-label classification.

3. Results and Discussion: Demonstrating the Potential of AiTLAS

In this section, we showcase the potential of the AiTLAS toolbox through a series of more detailed examples of its various functionalities and capabilities. We present five use cases that demonstrate the basic principles behind the toolbox, its modularity, and its flexibility. For each of the types of EO tasks that we highlighted earlier, we discuss every segment of the analysis pipeline. We start by loading a new dataset and performing exploratory data analysis, inspection, and pre-processing. Next, we demonstrate the use of a machine-learning model (provided in AiTLAS) for a given dataset as well as the evaluation of the model performance (through different evaluation measures, confusion matrices, and visualizations). Moreover, we show how users can utilize previously trained models

for making predictions on unseen images and quantitatively and qualitatively analyze the obtained results. Finally, we provide recipes for including new machine-learning architectures into the AiTLAS toolbox.

3.1. Image Classification

We start with a showcase of image scene classification. Note that the presented examples (and the obtained results) can be easily reproduced via a Jupyter notebook presented and further discussed in Appendix C. Moreover, an extensive analysis, performed using AiTLAS, of more than 500 DL models across a variety of multi-class and multi-label image classification tasks is presented in [3].

3.1.1. Data Understanding and Preparation

Currently, the `aitlas.datasets` module includes 22 ready-to-use datasets for EO image scene classification. To use these datasets, one needs to set the location of the images and a `csv` file with the labels for each image. The loaded images can then be transformed (i.e., via data augmentation), inspected, and visualized (including summaries over the complete dataset). In the following, we show the process of adding a new dataset and illustrate the capabilities for exploratory analysis.

For this purpose, we use the CLRS dataset [42] as a running example without loss of generality of the `aitlas.datasets` module. The CLRS dataset (available at https://github.com/lehaifeng/CLRS (accessed on 8 March 2023)) is designed for the task named continual/lifelong learning for EO image scene classification [42]. It comprises 15,000 remote sensing images covering over 100 countries divided into 25 scene classes. Each class is associated with 600 images with a size of 256 $\times$ 256 pixels and spatial resolution in the range of 0.26 m to 8.85 m.

Given that the dataset is shared without predefined splits (for training/validation/testing), we can first create them using the split task within the `aitlas.tasks` module. The user needs to only specify the data location and the ratios (in percentages) for the desired splits (e.g., here, we use 60/20/20 splits). The split task will output three separate `csv` files for each split, where each row in the `csv` denotes the relative path of the image (including the sub-folder name and the file name) and its label. For reproducibility, we provide further details in Appendix B and already prepared dataset (available at https://github.com/biasvariancelabs/aitlas-arena (accessed on 8 March 2023)).

Next, through the class *MultiClassClassificationDataset* from the `aitlas.datasets` module, one can load a new dataset as shown in Listing 1. This class also implements additional data transformations, which can be applied if necessary.

Listing 1. Loading an MCC dataset for remote sensing image scene classification using the *MultiClassClassificationDataset* class from the AiTLAS toolbox.

```
dataset_config = {
"data_dir": "/datasets/CLRS",
"csv_file": "/datasets/CLRS/images.csv" }
dataset = MultiClassClassificationDataset(dataset_config)
```

Loading a completely new dataset (beyond the ones currently supported within AiTLAS) can be achieved by explicitly defining a new data loader class that inherits from the class *MultiClassClassificationDataset*. Within the definition of the class, one needs to manually set the list of the labels/classes (with labels matching the labels provided from the `csv` file). One can also provide additional meta-data (such as name and URL) together with additional methods for data manipulation. Listing 2 provides a recipe for this using the CLRS dataset.

Listing 2. Adding a new MCC dataset in the AiTLAS toolbox.

```
from .multiclass_classification import MultiClassClassificationDataset

LABELS = ["airport", "bare-land", "beach", "bridge", "commercial",
"desert", "farmland", "forest", "golf-course", "highway",
"industrial", "meadow", "mountain", "overpass", "park",
"parking", "playground", "port", "railway", "railway-station",
"residential", "river", "runway", "stadium", "storage-tank"]

class CLRSDataset(MultiClassClassificationDataset):

url = "https://github.com/lehaifeng/CLRS"
labels = LABELS
name = "CLRS dataset"

def __init__(self, config):
super().__init__(config)
```

Once the dataset is ready, one can use methods from the `aitlas.visulizataion` module for data visualization and inspection. For instance, one can easily plot images from the dataset (as shown in Figure 6) and analyze their properties in terms of data distributions. A more detailed discussion of data exploration capabilities is given in Appendix C.

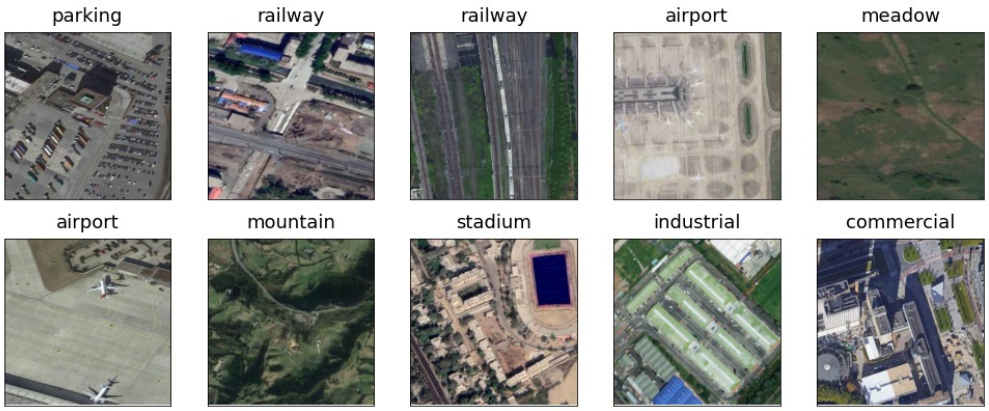


Figure 6. Example images with labels from the CLRS dataset.

3.1.2. Definition, Execution, and Analysis of a Machine Learning Pipeline

Given a dataset, we next focus on setting an approach that supports an image scene classification task. In this use case, we employ the Vision Transformer (ViT) model [76]. Specifically, we use a ViT with an input size of 224 × 224 and a patch resolution of 16 × 16 pixels. We showcase a pipeline with two variants: (i) a model "trained from scratch" using the CLRS dataset and (2) a pre-trained model on ImageNet-1K and then fine-tuned on the CLRS dataset. Note that the ViT model is trained/fine-tuned using the data splits we defined earlier.

We configure the model by setting several configuration parameters, i.e., the number of classes/labels, the learning rate, and the evaluation metrics. To use the pre-trained variant of the Vision Transformer (pre-trained on the ImageNet-1k dataset) in the configuration object, we also set the pre-trained parameter to *true*. We fine-tune the model on the CLRS dataset by calling the function `train_and_evaluate_model`. The code snippet for instantiating the model and running the training sequence is given in Listing 3.

Listing 3. Creating a model and executing model training.

```
epochs = 100
model_directory = "/experiments/CLRS"
model_config = {
"num_classes": 25,
"learning_rate": 0.0001,
"pretrained": True,
"metrics": ["accuracy", "precision", "recall", "f1_score"]}
model = VisionTransformer(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
epochs=epochs,
model_directory=model_directory,
val_dataset=validation_dataset,
run_id='1',)
```

To evaluate the learned model, we load the dataset's test split, set the evaluation metrics list, and run the evaluation sequence on the test data (Listing 4). Note that the predictive performance of the models in this setting is typically assessed by *top-n accuracy* score (typically n is set to 1 or 5) [65]. This score calculates the number of correctly predicted labels among the n most probable labels the model outputs. Besides accuracy, AiTLAS supports additional prediction performance metrics such as Macro Precision, Weighted Precision, Macro Recall, Weighted Recall, Macro F1 score, and Weighted F1 score, etc.

Listing 4. Evaluating a trained model using images from the test split.

```
test_dataset_config = {
"batch_size": 128,
"shuffle": False,
"data_dir": "/dataset/CLRS",
"csv_file": "/dataset/CLRS/test.csv",
"transforms": ["aitlas.transforms.ResizeCenterCropToTensor"]}
test_dataset = CLRSDataset(test_dataset_config)
model_path = "best_checkpoint.pth.tar"
model.metrics = ["accuracy", "precision", "recall", "f1_score"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

Finally, once the model has been trained and evaluated, users can analyze their performance. In this particular example, the ViT model that was first pre-trained on ImageNet-1K achieved an accuracy of 93.20%, substantially outperforming the counterpart trained from scratch, which obtained an accuracy of 65.47%. The performance can also be investigated via confusion matrices (Figure 7), allowing for a more fine-grained analysis of the model performance on individual classes. Finally, using AiTLAS, users can further analyze the predictions made by the model's Grad-CAM [118] activation maps. For instance, Figure 8 presents a sample output obtained from the ViT model, highlighting where the model focused when making the correct (or incorrect) predictions. This capability allows for further validation and diagnosis of the models.

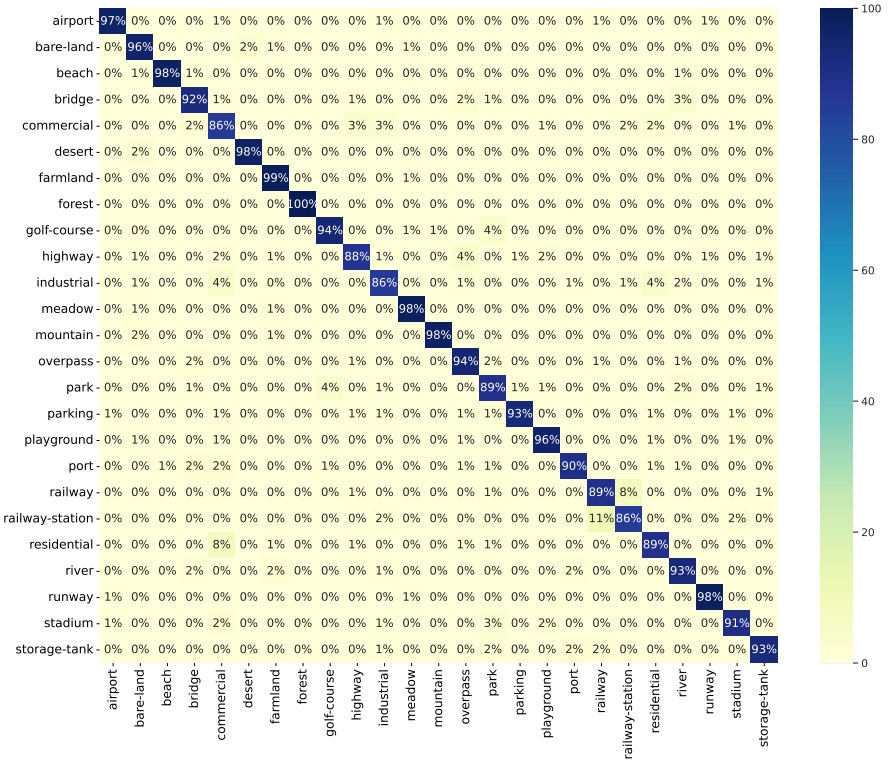


Figure 7. Confusion matrix obtained from a pre-trained Vision Transformer model applied on the CLRS dataset. The values denote percentages of correctly/incorrectly predicted labels.

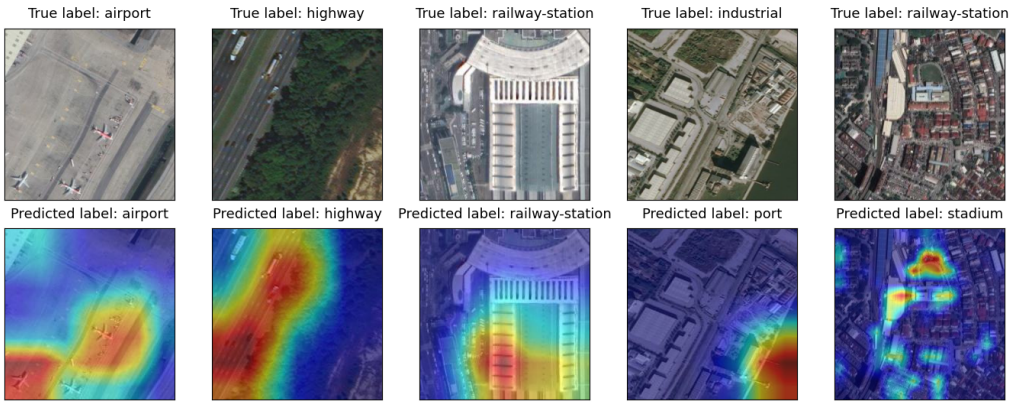


Figure 8. GradCAM visualizations for images, sampled from the CLRS dataset: (**top**) input images with their ground-truth label, (**bottom**) corresponding activation maps with predicted labels (from a ViT model).

A detailed discussion on the pipeline is presented in Appendix C, including templates for model learning, evaluation, and inspection, as well as guidelines on using the learned models for predicting tasks with unseen images.

3.2. Semantic Segmentation

In this section, we discuss the utility of the AiTLAS toolbox for semantic segmentation tasks. For this use case, we use the LandCover.ai (Land Cover from Aerial Imagery) dataset [119] and employ a DeepLabV3 model to perform the semantic segmentation task. The presented use case (together with the obtained results) can be easily reproduced using the Jupyter notebook presented and discussed in Appendix D.

3.2.1. Data Understanding and Preparation

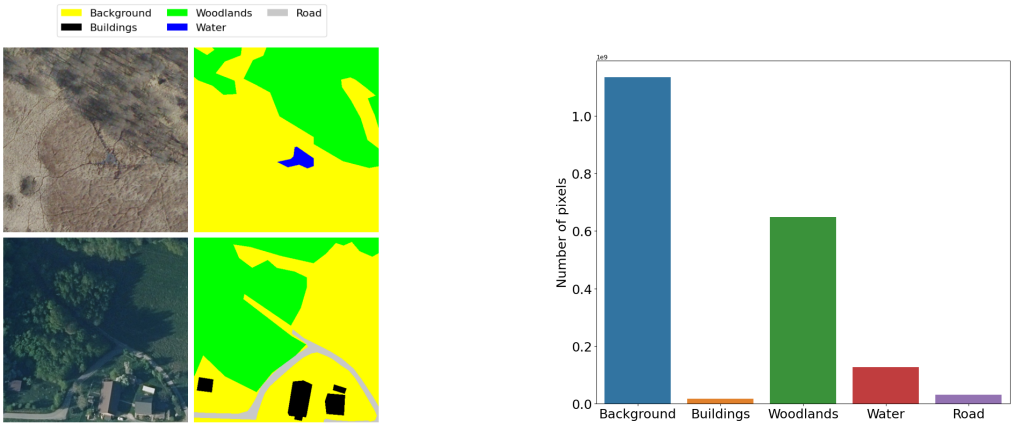
LandCover.ai (Land Cover from Aerial Imagery) (available at https://landcover.ai.linuxpolska.com/download/landcover.ai.v1.zip (accessed on 8 March 2023)) is a dataset for automatic mapping of buildings, woodlands, water, and roads from aerial images [119]. It contains a selection of aerial images taken over the area of Poland. The images have a spatial resolution of 25 or 50 cm per pixel with three spectral bands (RGB bands). The original 41 images and their corresponding masks are split into 512 × 512 tiles. The tiles are then shuffled and organized into 70%/15%/15% of the tiles for training, validation, and testing, respectively.

AiTLAS implements data loaders for creating, loading, and preparing datasets for semantic segmentation. Specifically, the class *SemanticSegmentationDataset* from the `aitlas.datasets` module is the base class for creating a dataset, which loads images and the corresponding segmentation masks from a `csv` file. For example, in this case, this is performed within the instanced *LandCoverAiDataset* (presented in Listing 5), which sets the labels and their color mapping (used only for visualization as presented in Figure 9). A complete example of the data inspection capabilities is given in Appendix D.

Listing 5. Adding a new dataset for semantic segmentation in the AiTLAS toolbox.

```
class LandCoverAiDataset(SemanticSegmentationDataset):
url = "https://landcover.ai.linuxpolska.com/"
labels = ["Background", "Buildings", "Woodlands", "Water", "Road"]
color_mapping = [[255, 255, 0], [0, 0, 0], [0, 255, 0], [0, 0, 255], [200, 200,
    200]]
name = "Landcover AI"

def __init__(self, config):
super().__init__(config)
```



(**a**) (**b**)

Figure 9. Example images with masks from the LandCover.ai dataset. Each pixel is labeled with one of the following labels: background (yellow), buildings (black), woodlands (green), water (blue) and road (gray) (**a**) and pixel distribution within the labels (**b**).

3.2.2. Definition, Execution and Analysis of a Machine Learning Pipeline

Once the dataset has been appropriately set, we focus on setting, training, and evaluating the models. For this use case, we train two variants of DeepLabV3 models: (i) one "trained from scratch" using only the LandCover.ai dataset, and (2) a model with pre-trained weights on a subset of the COCO dataset [120] (using only the 20 categories also present in the Pascal VOC dataset [66]), subsequently fine-tuned on LandCover.ai dataset. Listing 6 presents the AiTLAS configuration for setting and executing the training procedure. We use mean intersection over union ($mIoU$) as an evaluation measure, which denotes the area of the overlap between the ground truth and predicted label divided by the total area, averaged across the different labels.

Listing 6. Creating an instance of a DeepLabv3 model and executing model training.

```
epochs = 100
model_directory = "/experiments/landcoverai"
model_config = {
"num_classes": 5,
"learning_rate": 0.0001,
"pretrained": True,
"threshold": 0.5,
"metrics": ["iou"] }

model = DeepLabV3(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
val_dataset=validation_dataset,
epochs=epochs,
model_directory=model_directory,
run_id='1')
```

Once the training routine has finished, evaluating the model can be performed as presented in Listing 7.

Listing 7. Evaluating a trained DeepLabv3 model using images from the test split.

```
test_dataset_config = {
"batch_size": 4,
"shuffle": False,
"data_dir": "/dataset/landcoverai/images",
"csv_file": "/dataset/landcoverai/test.txt",
"transforms": ["aitlas.transforms.MinMaxNormTranspose"],
"target_transforms": ["aitlas.transforms.Transpose"]
}

test_dataset = LandCoverAiDataset(test_dataset_config)
model_path = "/experiments/landcoverai/best_checkpoint.pth.tar"
model.metrics = ["iou"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

The results presented in Table 7 summarize the segmentation performance of the trained models. In this example, both DeepLabv3 models resulted in a similar performance, with the pre-trained model having a (practically) insignificantly better performance than its counterpart trained from scratch. In general, the values of the *IoU* for the labels 'Road' and 'Building' are lower than the other labels since they are usually narrow (roads) and/or often small (buildings), thus typically challenging for segmentation models. Running additional experiments with different model architectures, setups, and datasets is straightforward within AiTLAS, requiring only simple modifications to the presented routines. Additional templates for this task, including additional visualizations and application of the model to external images, are given in Appendix D.

Table 7. Label-wise $IoU(\%)$ and $mIoU$ for the DeepLabv3 models trained on the LandCover.ai dataset.

Model/Label	Background	Buildings	Woodlands	Water	Road	mIoU	Training Time
Trained from scratch	93.813	80.304	91.952	94.877	69.190	86.027	4.5 h
Pre-trained on COCO	93.857	80.650	91.964	95.145	68.846	86.093	5 h

3.3. Object Detection

We next discuss using the AiTLAS toolbox for object detection. In this use case, we show using a Faster R-CNN model on the HRRSD dataset [52]. All of the necessary details of the developed resources are further given in Appendix E.

3.3.1. Data Understanding and Preparation

The AiTLAS toolbox supports the representation of the data for the object detection task through its `aitlas.datasets` module via the base classes *ObjectDetectionPascalDataset* and *ObjectDetectionCocoDataset*. They implement the two most widely used data representation formats for object detection: PascalVOC [121] and COCO [120], respectively. More specifically, the Pascal VOC format includes an XML file for each image in the dataset containing information about the bounding boxes of the objects present in the image, together with some additional metadata such as category/label, level of difficulty, and an indicator of whether the object is truncated (partially visible). On the other hand, the COCO annotation format stores the annotations in JSON files for the training, testing, and validation parts of the data. The JSON file contains a list of each object annotation from every image in the dataset. The annotation includes coordinates for the bounding box, the area of the bounding box, category/label, and an 'iscrowd' indicator for the number of objects in an image (which is 0 for single objects or 1 for a collection of objects).

The HRRSD (available at https://github.com/CrazyStoneonRoad/TGRS-HRRSD-Dataset (accessed on 8 March 2023)) [52,122] dataset contains 21,761 color images acquired from Google Earth with spatial resolution ranging from 0.15 to 1.2 m, and 4961 color images acquired from Baidu Maps with a spatial resolution ranging from 0.6 to 1.2 m. The dataset is divided into a training portion (5401 images), a validation portion (5417 images), and a test portion (10,943 images). An image from the dataset may contain several objects or just one and may contain objects from the 13 different categories/labels. The total number of object instances is 55,740.

Similarly to the use cases presented earlier, loading a dataset into AiTLAS involves instantiating *ObjectDetectionPascalDataset*, as shown in Listing 8. The class *ObjectDetectionPascalDataset* implements additional functionalities for further inspection of the loaded data. For example, one can visualize images from the dataset (as shown in Figure 10) coupled with the number of instances for each category within the dataset.

Listing 8. Loading HRRSD dataset using the class *ObjectDetectionPascalDataset* from the AiTLAS toolbox.

```
dataset_config = {
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/train.txt", }

dataset = ObjectDetectionPascalDataset(dataset_config)
```

Figure 10. Example images with bounding boxes for the objects from the HRRSD dataset.

3.3.2. Definition, Execution and Analysis of a Machine Learning Pipeline

As mentioned earlier, in this use case, we use the Faster R-CNN model with a ResNet-50-FPN backbone [84]. Specifically, here we also train and evaluate two variants of the model: "trained from scratch" using the HRRSD dataset and a pre-trained model on the COCO dataset [66] and then fine-tuned on the HRRSD dataset. The code snippet for

creating and training these models is shown in Listing 9. To evaluate the models, one needs to create a configuration object for the training split of the HRRSD data, load the data, set the path to the trained model, and run the evaluation process (as shown in Listing 10). As an evaluation measure, we use the mean average precision (mAP) as defined in the Pascal VOC Challenge [121], computed as the average precision value taken at recall values ranging from 0 to 1 (i.e., the area under the precision/recall curve) and then averaged over all classes. The performance of object detection models can also be evaluated using IoU between the predicted and ground-truth bounding boxes.

Listing 9. Creating an instance of a Faster R-CNN model and executing model training.

```
epochs = 100
model_directory = "/experiments/hrrsd"
model_config = {
"num_classes": 14,
"learning_rate": 0.0001,
"pretrained": True,
"threshold": 0.5,
"metrics": ["map"] }

model = FasterRCNN(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
val_dataset=validation_dataset,
epochs=epochs,
model_directory=model_directory,
run_id='1'
)
```

Listing 10. Testing a trained model with images from the test split.

```
test_dataset_config = {
"batch_size": 4,
"shuffle": False,
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/test.txt",
"joint_transforms": ["aitlas.transforms.ResizeToTensorV2"] }

test_dataset = ObjectDetectionPascalDataset(test_dataset_config)
model_path = "/experiments/hrrsd/best_checkpoint.pth.tar"
model.metrics = ["map"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

The results of this particular use case show that the pre-trained Faster R-CNN model leads to an mAP of 81.436%, outperforming the variant trained from scratch with an mAP of 77.412%. Further investigation shows that, in this case, the most challenging objects to detect are 'Crossroad' and 'T Junction' (due to the similarity of both objects). AiTLAS allows for further quantitative analysis of these results by examining the predicted outputs (images with the detected objects and the categories/labels for each object) as shown in Figure 11. Further details for this use case are presented in Appendix E.

Figure 11. Example images with the predicted bounding boxes and object labels ('ship', 'T junction' and 'airplane', respectively) using a Faster R-CNN model.

3.4. Crop Type Prediction

For our last use case, we show the capabilities of the AiTLAS toolbox on the task of crop type prediction. For this purpose, we train and evaluate an LSTM model applied to the AiTLAS NLD dataset [63]. Merdjanovska et al. [63] present an extensive analysis of this task, performed using the AiTLAS toolbox, comparing the performance of several state-of-the-art deep-learning architectures. All of the developed resources for this use case are also presented and discussed in Appendix F.

3.4.1. Data Understanding and Preparation

Typically, the data format for crop-type prediction tasks is different compared to data for the other EO tasks because of their temporal component that needs to be taken into consideration. The AiTLAS toolbox supports crop-type prediction datasets via the base class *CropsDataset* and the *EOPatchCrops* class (a wrapper for working with *EOPatches* [23]). The EOPatch format stores multi-temporal remotely sensed data of a single patch of the Earth's surface as constrained by the bounding box in a given coordinate system. The patch can be a rectangle, polygon, or pixel in space. The same object can also be used to store derived measures and indices from the patch, such as means, standard deviations, etc.

The AiTLAS NLD dataset [63] consists of Sentinel 2 data, resampled at 10-day intervals in the periods of March–November of 2017, 2018, and 2019 (resulting in 28 distinct dates for each year). We use images from 10 m and 20 m bands which contain eight spectral bands: B3, B4, B5, B6, B7, B8, B11, and B12. Additionally, the dataset contains three calculated indices: NDVI (normalized difference vegetation index), NDWI (normalized difference water index), and brightness (euclidean norm). Each polygon observation describes the temporal profile of a crop field and is associated with multivariate time series obtained by averaging the reflectance values at a crop-field level extracted from the Sentinel 2 data. In this way, each polygon is represented as a two-dimensional vector: the first dimension represents the (11) spectral bands, and the second one the (28) time steps. The crop type data categorization was created from the Land Parcel Identification System (LPIS). Each crop type label describes one crop field (or parcel), which in turn is identified with a polygon border. Figure 12 shows an example of the different field geometry present in the data.

Figure 12. Example field geometries for the AiTLAS NLD dataset. The different colors of the polygons in the sample represent different crop types.

The loading configuration (shown on Listing 11) of such data includes mapping the path to the data, an index file (a separate csv file) which contains the class mappings of each polygon/patch, as well as the train/validation/test data splits (in terms of regions) for running the ML pipeline. Users can also specify other attributes for working with datasets, such as batch size, data shuffling, and the number of workers.

Listing 11. Loading the AiTLAS NLD dataset.

```
dataset_config = {
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"batch_size": 128,
"shuffle": True,
"num_workers": 4,
"regions":["train", "test", "val",],
}
dataset = EOPatchCrops(dataset_config)
```

3.4.2. Definition and Execution of Machine Learning Tasks

AiTLAS casts the task of crop type prediction as a multi-class classification task, defined with the *BaseMulticlassClassifier* class, within the `aitlas.base` module. To illustrate the use of AiTLAS for crop type prediction, we use an LSTM model [87]. Specifically, we perform experiments on the AiTLAS NLD dataset independently for each of the three years. The goal is to examine the models' behavior and performance and how it varies across the years. For measuring the models' predictive performance, we use standard metrics such as accuracy, weighted F1 score, and Kappa coefficient. In the context of AiTLAS, initializing and creating the training routine is straightforward: one needs to set the model parameters in the configuration object, instantiate the model, and run the training sequence (as shown in Listing 12).

Listing 12. Creating an instance of an LSTM model and executing model training.

```
epochs = 100
model_directory = "./experiments/LSTM"
model_config = {
"input_dim":11,
"num_classes": 10,
"learning_rate": 0.001,
"dropout" : 0.2,
"weight_decay": 0.0001,
"metrics":["accuracy","f1_score", "kappa"] }
model = LSTM(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
epochs=epochs,
model_directory=model_directory,
val_dataset=validation_dataset,
run_id='1',)
```

Once the model is trained, one can evaluate the model in a predictive setting with unseen data (Listing 13). In this example, the trained LSTM model shows consistent performance (across the three datasets/years) with accuracy in the range of ∼84–85% and a weighted F1 score in the range ∼82–84%. AiTLAS allows for more fine-grained per-label analysis of the model performance. For instance, in this example, the F1 score for *Temporary grasses and grazings* in 2017 is 45.23, while in 2018 raises to 59.34. Such insights can further help diagnose and improve the model's performance. Finally, AiTLAS also supports qualitative prediction analysis through visualizations of the predicted regions (and their labels). Further details of this use case, including complete results of the experiments, are given in Appendix F.

Listing 13. Evaluating a trained LSTM model.

```
labels = ["Permanent grassland", "Temporary grasses and grazings", "Green maize",
    "Potatoes (including seed potatoes)",
"Common winter wheat and spelt", "Sugar beet (excluding seed)", "Other farmland",
    "Onions",
"Flowers and ornamental plants (excluding nurseries)", "Spring barley",]

test_dataset_config = {
"batch_size": 32,
"shuffle": False,
"num_workers": 4,
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"regions":["test", ],}

test_dataset = EOPatchCrops(test_dataset_config)
y_true, y_pred, y_prob = model.predict(dataset=test_dataset,)

eopatches_path = "/home/user/data/CropTypeNetherlands/2019/eopatches/"
patch = "eopatch_7495"
```

3.5. Adding a New Machine Learning Model in AiTLAS

In the use cases discussed previously, we showcased the capability of AiTLAS with the already available methods and model architectures. However, AiTLAS is modular and easily extensible to new approaches. Here we present a template for adding novel model architectures into the AiTLAS toolbox. This includes instantiating from one of the model base classes from the `aitlas.base` module, which corresponds to a particular EO task. For instance, adding a new model for image scene classification, such as *EfficientNetV2* as implemented in the PyTorch [66] model catalog. The model can be added by creating an inherited class from *BaseMulticlassClassifier*, which already implements all the requirements to train a deep-learning model successfully.

Namely, the implementation (e.g., Listing 14) includes initialization of the model variable from the base class and overriding the forward function. In the `init` function, the model variable is initialized using the `efficientnet_v2_m` function, which constructs the EfficientNetV2-M architecture introduced by Tan and Le [123]. Users can also include pre-trained model variants by setting the variable 'pretrained', which will lead to setting a pre-trained model from the ImageNet-1K dataset. The newly added *EfficientNetV2* model can then be used for remote sensing image scene classification in a similar manner as the use case presented in Section 3.1.

Listing 14. Adding EfficientNetV2 as a new model for remote sensing image scene classification in the AiTLAS toolbox.

```
import torchvision.models as models
import torch.nn as nn

from ..base import BaseMulticlassClassifier, BaseMultilabelClassifier

class EfficientNetV2(BaseMulticlassClassifier):
name = "EfficientNetV2"

def __init__(self, config):
super().__init__(config)
if self.config.pretrained:
self.model = models.efficientnet_v2_m(
weights=models.EfficientNet_V2_M_Weights.IMAGENET1K_V1,
progress=False
)
in_features = self.model.classifier[1].in_features
self.model.classifier[1] = nn.Linear(
in_features, self.config.num_classes
)
else:
self.model = models.efficientnet_v2_m(
weights=None, progress=False,
num_classes=self.config.num_classes
)

def forward(self, x):
return self.model(x)
```

4. Conclusions

We present AiTLAS (https://aitlas.bvlabs.ai (accessed on 8 March 2023)), an open-source, state-of-the-art toolbox for exploratory and predictive analysis of satellite imagery. AiTLAS is a versatile Python library applicable to a variety of different tasks from EO, such as image scene classification, image segmentation, object detection, and crop type prediction (time series classification) tasks. It provides the means for straightforward construction and execution of complete end-to-end EO pipelines catering to users' needs with different goals, domain backgrounds, and levels of expertise.

From an EO perspective, where users typically focus on a particular application, AiTLAS supports building complete data analysis pipelines starting from data preparation and understanding of the data, through leveraging state-of-the-art deep-learning architectures for various predictive tasks, to quantitative and qualitative analysis of the predicted outcomes. As such, the capabilities of AiTLAS expand significantly in comparison to other related libraries such as eo-learn [23], OTB [26], and CANDELA [28]. Namely, in addition to data handling, AiTLAS implements approaches for model learning, be they training models from scratch or employing/fine-tuning pre-trained models freely available via the AiTLAS model catalog. Moreover, AiTLAS provides an additional, more comprehensible, configuration layer for executing EO data-analysis pipelines that do not require significant familiarity with the underlining machine-learning technologies. We believe this capability substantially flattens the learning curve for constructing and executing novel EO data anal-

ysis pipelines. From an AI perspective, AiTLAS implements the necessary components for implementing novel methods for EO data analysis, both in terms of (novel) deep-learning architectures as well as approaches for data handling/transformation. Moreover, it further facilitates the development of new approaches by providing easy access to formalized AI-ready data (via the AiTLAS EO data catalog) and their evaluation via standardized and extensive benchmark framework [3].

The main motivation of AiTLAS is bringing together the AI and EO communities by providing extensible, easy-to-use, and, more importantly, open-source resources for EO data analysis. The design principles of AiTLAS, showcased in this work, build on this motivation by providing:

1. *User-friendly, accessible, and interoperable* resources for data analysis through easily configurable and readily usable pipelines. The resources can be easily adapted by adjusting the configuration (JSON) files to a specific task at hand.
2. *Standardized, verifiable, and reusable* data handling, wrangling, and pre-processing approaches for constructing AI-ready data. The AiTLAS datasets are readily available for use through the toolbox and accessible through its EO data catalog (http://eodata.bvlabs.ai) , which incorporates FAIR [124] ontology-based semantic (meta) data descriptions of AI-ready datasets;
3. *Modular and configurable* modeling approaches and (pre-trained) models. The implemented approaches can be easily adjusted to different setups and novel analysis pipelines. Moreover, AiTLAS includes the most extensive open-source catalog of pre-trained EO deep-learning models (currently with more than 500 models) that have been pre-trained on a variety of different datasets and are readily available for practical applications.
4. *Standardized and reproducible* benchmark protocols (for data and models) that are essential for developing trustworthy, reproducible, and reusable resources. AiTLAS provides the resources and the necessary mechanisms for reconciling these protocols across tasks, models (and model configurations), and processing details of the datasets being used.

The development of AiTLAS is an ongoing effort. We foresee its evolution in three primary directions. First, it will continue to keep pace with the growing body of resources developed by the AI4EO community, continuously extending the catalogs of available AI-ready EO datasets and novel mode architectures. Second, and more application focused, we will continue the development of templates for specific use cases and pipelines tailored for different domains (such as agriculture, urban planning, geology, archaeology, etc.). Third, we will focus on extending the AiTLAS capabilities with self-supervised learning (SSL) approaches as a response to various practical challenges akin to costly and tedious labeling processes of large amounts of unlabeled data [125]. We believe that such extensions will bring many practical benefits in different downstream applications [99,126–128], which will undoubtedly further increase the utility of AiTLAS.

Author Contributions: Conceptualization, I.D., N.S. and D.K.; methodology, I.D., I.K. N.S. and D.K.; software, I.D. and I.K.; validation, I.D., I.K. and N.S.; formal analysis, I.D., I.K., N.S. and D.K.; investigation, I.D., I.K., N.S. and D.K.; resources, I.D., I.K., N.S.; data curation, A.K. and P.P. ; writing—original draft preparation, I.D., I.K. and D.K.; writing—review and editing, A.K., P.P., N.S. ; visualization, I.D. and N.S.; supervision, D.K.; project administration, D.K.; funding acquisition, N.S., P.P. and D.K. All authors have read and agreed to the published version of the manuscript.

Funding: The AiTLAS toolbox is developed within the grant from the European Space Agency (ESRIN): AiTLAS–Artificial Intelligence toolbox for Earth Observation (ESA RFP/3-16371/19/I-NB) awarded to Bias Variance Labs, d.o.o.

Data Availability Statement: AiTLAS toolbox is available at https://aitlas.bvlabs.ai (accessed on 8 March 2023), together with the accompanying results and the Jupyter notebooks from the use cases. The datasets used in the use cases are described and available at the AiTLAS EO data repository http://eodata.bvlabs.ai (accessed on 8 March 2023). Further benchmark experiments, including pre-

trained models for image scene classification, are available at https://github.com/biasvariancelabs/aitlas-arena (accessed on 8 March 2023).

Acknowledgments: We thank Žiga Kokalj, Sveinung Loekken, Sara Aparicio, Bertrand Le Saux, Elena Merdjanovska, Stefan Popov, Stefan Kalabakov and Sašo Džeroski—for their useful feedback and support during the development of AiTLAS. We also thank Sofija Dimitrovska for her thoughtful feedback on the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Third-Party Dependencies

Table A1. Third-party libraries used by AiTLAS described with their scope of usage and purpose within our toolbox.

Library	Scope of Usage	Purpose
PyTorch Vision [66]	`aitlas.models`	Pre-built deep-learning model architectures.
PyTorch Image Models [72]	`aitlas.models`	Pre-built deep-learning model architectures.
Segmentation Models PyTorch [79]	`aitlas.models`	Pre-built deep-learning model architectures specifically for segmentation tasks.
Albumentations [129]	`aitlas.transforms`	Applying transormations and augmentations.
NumPy [130]	`aitlas.base`	General scientific computing
Scikit-learn [131]	`aitlas.base`	Used for metric calculations
Scikit-multilearn [132]	`aitlas.tasks`	Used for stratified dataset splitting.
Seaborn [133]	`aitlas.visualizations`	For visualizations.
Matplotlib [133]	`aitlas.visualizations`	For visualizations
TensorBoard [27]	`aitlas.base`	Enable logging the train/validation loss during model training as well as any other supported metrics. This then allows for those statistics to be visualized in the *TensorBoard* UI.
zipp [134]	`aitlas.utils`	Enables working with zip files.
dill [135]	`aitlas.utils`	Extends Python's pickle module for serializing and de-serializing Python objects to the majority of the built-in data types.
lmdb [136]	`aitlas.datasets`	Enables working with LMDB data.
tifffile [137]	`aitlas.utils`	Reads TIFF files from any storage.
h5py [138]	`aitlas.datasets`	Provides an interface to the HDF5 binary data format.
click [139]	`aitlas.base`	Enables creating command line interfaces.
munch [140]	`aitlas.base`	Provides attribute-style access to objects.
marshmallow [141]	`aitlas.base`	It is an ORM/ODM/framework-agnostic library for converting complex data types to and from native Python data types.
Pytoch Metrics [142]	`aitlas.metrics`	Utility library used for computing performance metrics.

Appendix B. Split Task within the AiTLAS Toolbox for Creating Train, Validation and Test Splits

When constructing a data analysis pipeline, it is common to split the available data into separate sets/splits that can be used for model training, validation, and testing. While the training split is used for training the ML model, the validation split is used for estimating the optimal hyper-parameters of the model. The test split is then used to evaluate the model's performance on new, unseen data. In the context of remote sensing image scene classification, the AiTLAS toolbox provides an interface for performing the splitting task to generate these data splits. The task is very convenient as most of the datasets are published without these splits, and it is up to the practitioner to define the splits.

The split task in the AiTLAS toolbox supports sampling strategies for generating the splits: random and stratified. The latter method ensures that the distribution of the target/class variable(s) is the same among the different splits [143]. The configuration file that can be used to run the AiTLAS toolbox and obtain the required splits for a given dataset is provided in Listing A1.

Listing A1. Configuration file for creating train, validation and test splits.

```
{
"task": {
"classname": "aitlas.tasks.StratifiedSplitTask",
"config": {
"split": {
"train": {
"ratio": 60,
"file": "./data/CLRS/train.csv"
},
"val": {
"ratio": 20,
"file": "./data/CLRS/val.csv"
},
"test": {
"ratio": 20,
"file": "./data/CLRS/test.csv"
}
},
"data_dir": "./data/CLRS"
}
}
}
```

An example `csv` file generated using the split task is given in Listing A2.

Listing A2. Example `csv` file with the required format by the AiTLAS toolbox for remote sensing image scene classification datasets.

```
airport/airport_321_Level1_0.53m.tif,airport
storage-tank/storage-tank_489_Level1_0.50m.tif,storage-tank
golf-course/golf-course_374_Level3_2.05m.tif,golf-course
parking/parking_561_Level1_0.39m.tif,parking
residential/residential_77_Level1_0.49m.tif,residential
meadow/meadow_5_Level3_1.37m.tif,meadow
mountain/mountain_538_Level3_7.60m.tif,mountain
mountain/mountain_180_Level2_3.66m.tif,mountain
```

In the configuration file, the split task has to be set. In Listing A1, in order to perform a stratified splitting of the data, we have chosen the `aitlas.tasks.StratifiedSplitTask`. Then, we include the path to the folder in which the images are stored. Additionally, for the MLC dataset, a `csv` file containing one-hot encoded classes or labels must be provided alongside the images. However, for the MCC dataset, a `csv` file is not necessary since the images for different classes or labels are located in separate sub-folders within the main or root folder. We use ratios to generate the splits required for training, validation, and testing, used to determine the proportion of images in each split. Based on these ratios, a split-task automatically generates a `csv` file that contains the image names for each split. Additional configuration files for different datasets can be found in the AiTLAS repository (https://github.com/biasvariancelabs/aitlas/blob/master/examples/ (accessed on 8 March 2023)).

Appendix C. Remote Sensing Image Scene Classification

A Jupyter Notebook for demonstrating remote sensing image scene classification is available in the AiTLAS repository (https://github.com/biasvariancelabs/aitlas/blob/master/examples/multiclass_classification_example_clrs.ipynb (accessed on 8 March 2023)). The notebook contains a step-by-step sample code for running an image scene classification task. More specifically, we demonstrate the standard steps on how to load and split the data and examine its label distribution. It also showcases the steps for training and evaluation of the model and visualizing the prediction.

We use the CLRS multi-class dataset, which consists of 25 land cover classes. To obtain the dataset, it first has to be downloaded from the repository (https://github.com/lehaifeng/CLRS (accessed on 8 March 2023) and unzipped, after which we obtain the data organized in subfolders containing images for each label/class. However, since the

CLRS dataset does not come with predefined train, validation, and test splits, the split task defined within the AiTLAS toolbox is used to generate the splits. More details about the split task can be found in Appendix B.

Next, we create an instance of the CLRSDataset class from the toolbox. To create the instance, we provide the folder with the images and a `csv` file with a list of images and labels. Additionally, we set the batch size for the data loader, the shuffle parameter to reshuffle the data at each epoch, and we set the number of workers/sub-processes to load the data. Listing A3 provides a code snippet illustrating how to create the instance. The created instance loads the image data from the dataset and offers additional functionalities for inspection and visualization.

Listing A3. Load a dataset using the AiTLAS toolbox, inspect images and calculate the class distribution.

```
dataset_config = {
"data_dir": "/datasets/CLRS",
"csv_file": "/datasets/CLRS/train.csv",
"batch_size": 128,
"shuffle": True,
"num_workers": 4
}
dataset = CLRSDataset(dataset_config)
fig = dataset.show_image(340)
dataset.data_distribution_table()
```

We can use the `show_image` function to display images from the dataset. To inspect the label distribution in the dataset, we can use the `data_distribution_table` function. The class distribution of the train, test, and validation splits can be calculated and presented as shown in Figure A1.

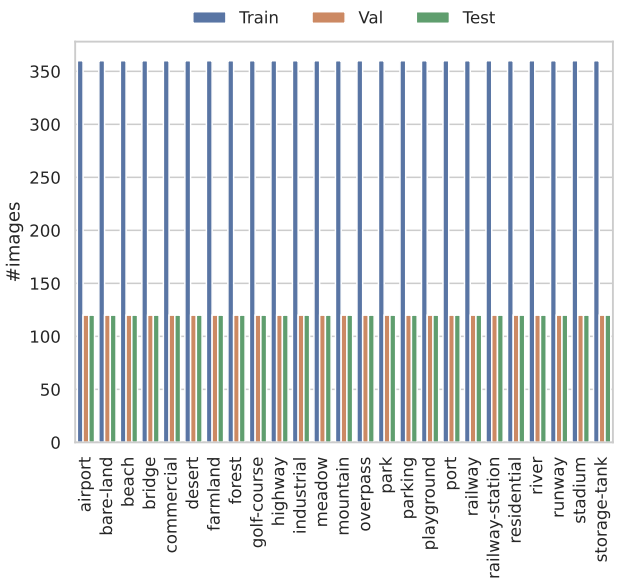


Figure A1. Class distribution for the CLRS dataset across the training, validation, and testing splits of the data.

In the next step, we continue specifying the model learning task by providing the training and validation data and a deep-learning model. The code snippet for creating train and validation datasets is given in Listing A4. Additionally, Listing A4 shows an example of specifying the data transformation parameter. Transformations are used to process the data to make it suitable for training. They can also augment the data to represent a more comprehensive set of possible data points, resulting in better performance and model generalization to address overfitting. Here we give an example of applying the *ResizeRan-*

domCropFlipHVToTensor on the training data, which first resizes all the images to 256 × 256, selects a random crop of size 224 × 224, and then applies random horizontal and/or vertical flips. On the test data, we apply the *ResizeCenterCropToTensor* transformation, which resizes the images to 256 × 256 and then applies a central crop of size 224 × 224.

Listing A4. Load the training and validation dataset.

```
train_dataset_config = {
"batch_size": 128,
"shuffle": True,
"data_dir": "/datasets/CLRS",
"csv_file": "/datasets/CLRS/train.csv",
}

train_dataset = CLRSDataset(train_dataset_config)
train_dataset.transform = ResizeRandomCropFlipHVToTensor()

validation_dataset_config = {
"batch_size": 128,
"shuffle": False,
"data_dir": "/datasets/CLRS",
"csv_file": "/datasets/CLRS/val.csv",
"transforms": ["aitlas.transforms.ResizeCenterCropToTensor"]
}

validation_dataset = CLRSDataset(validation_dataset_config)
```

For learning, we use the Vision Transformer model, available in the toolbox. We configure the model by setting several configuration parameters, i.e., the number of classes/labels, the learning rate, and the evaluation metrics. To use the pre-trained variant of the Vision Transformer (pre-trained on the ImageNet-1k dataset) in the configuration object, we also set the pre-trained parameter to *true*. We fine-tune the model on the CLRS dataset by calling the function `train_and_evaluate_model`. The code snippet for instantiating the model and running the training sequence is given in Listing A5.

Listing A5. Creating a model and start of model training.

```
epochs = 100
model_directory = "/experiments/CLRS"
model_config = {
"num_classes": 25,
"learning_rate": 0.0001,
"pretrained": True,
"metrics": ["accuracy", "precision", "recall", "f1_score"]
}
model = VisionTransformer(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
epochs=epochs,
model_directory=model_directory,
val_dataset=validation_dataset,
run_id='1',
)
```

We use *ReduceLROnPlateau* as a learning scheduler—it reduces the learning rate when the loss has stopped improving. Namely, models often benefit from reducing the learning rate by a factor once learning stagnates: ReduceLROnPlateau tracks the values of the loss measure, reducing the learning rate by a given factor when there is no improvement for a certain number of epochs (denoted as 'patience'). In our experiments, we track the value of the validation loss with patience set to 5 and a reduction factor set to 0.1 (the new learning rate will thus be $lr * factor$). The maximum number of epochs is set to 100. We also apply early stopping criteria if no improvements in the validation loss are observed over 10 epochs. The best checkpoint/model found (with the lowest validation loss) is saved and then applied to the test part to obtain the final assessment of the predictive performance.

To evaluate the learned model, we load the test split of the dataset, set the list of evaluation metrics, and run the evaluation sequence on the test data (Listing A6). The predictive performance of the models for MCC is typically assessed by reporting the *top-n accuracy* score (typically n is set to 1 or 5) [65]. This score is calculated by checking whether the correct label is placed among the n most probable labels outputted by the model. In this use case, we report *top-1 accuracy*, denoted as 'Accuracy', Macro Precision, Weighted Precision, Macro Recall, Weighted Recall, Macro F1 score, and Weighted F1 score. Note that since for MCC tasks, the micro-averaged measures such as F1 score, Micro Precision, and Micro Recall have values equal to accuracy, we do not report them separately.

Listing A6. Testing the model using the images from the test split.

```
test_dataset_config = {
"batch_size": 128,
"shuffle": False,
"data_dir": "/dataset/CLRS",
"csv_file": "/dataset/CLRS/test.csv",
"transforms": ["aitlas.transforms.ResizeCenterCropToTensor"]
}

test_dataset = CLRSDataset(test_dataset_config)
model_path = "best_checkpoint.pth.tar"
model.metrics = ["accuracy", "precision", "recall", "f1_score"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

The results from the ViT models trained from scratch, pre-trained on ImageNet-1K, and then fine-tuned on the specific dataset are given in Table A2. From the presented results, it is evident that leveraging pre-trained models can lead to significant performance improvements on image classification tasks [144], and in particular on tasks in EO domains [145]. The results also show the average training time per epoch, the total training time, and the epoch in which the lowest value for the validation loss has been obtained.

Table A2. Detailed results for the ViT models trained on the CLRS dataset.

Model/Metric	Accuracy	Macro Precision	Weighted Precision	Macro Recall	Weighted Recall	Macro F1 score	Weighted F1 Score	Avg. Time/Epoch (s)	Total Time (s)	Best Epoch
Trained from scratch	65.47	66.41	66.41	65.47	65.47	65.49	65.49	24.96	1173	32
Pre-trained on ImageNet-1K	93.20	93.29	93.29	93.20	93.20	93.22	93.22	25.32	785	21

The performance of the ViT model for the individual classes from the CLRS dataset can be analyzed from the results presented in Table A3 and Figure 7. Table A3 gives the precision, recall, and F1 score at a class level for the pre-trained ViT model on the CLRS dataset, and Figure 7 shows the confusion matrix. We can note that the F1 score of most classes is over 90%. The confusion matrix shows the effect of class similarity for the classes 'railway' and 'railway-station' to the overall performance—the model has difficulty discerning between these two classes.

Table A3. Per class results for the pre-trained Vision Transformer model on the CLRS dataset.

Label	Precision	Recall	F1 Score
airport	97.48	96.67	97.07
bare-land	92.00	95.83	93.88
beach	99.15	97.50	98.32
bridge	90.91	91.67	91.29
commercial	79.84	85.83	82.73
desert	97.50	97.50	97.50
farmland	93.70	99.17	96.36
forest	100.00	100.00	100.00
golf-course	94.96	94.17	94.56
highway	92.11	87.50	89.74
industrial	88.79	85.83	87.29
meadow	96.72	98.33	97.52
mountain	99.15	97.50	98.32
overpass	89.68	94.17	91.87
park	85.60	89.17	87.35
parking	98.25	93.33	95.73
playground	95.04	95.83	95.44
port	94.74	90.00	92.31
railway	86.29	89.17	87.70
railway-station	88.79	85.83	87.29
residential	90.68	89.17	89.92
river	90.32	93.33	91.80
runway	98.33	98.33	98.33
stadium	95.61	90.83	93.16
storage-tank	96.55	93.33	94.92

Additionally, the learned model can be used for predicting the labels of unseen images using the `predict_image` function. The function takes the image, the labels, and the transformation instance as arguments and returns the predicted class and the confidence score of the prediction for the given image. The output of this function is shown in Figure A2. The code snippet to obtain the prediction for a given image is shown in Listing A7.

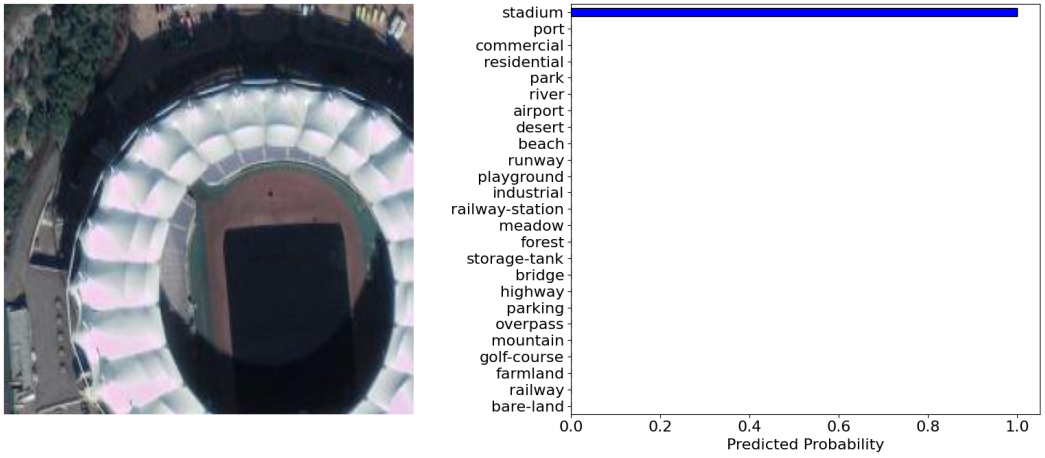


Figure A2. Example image with the predicted class and probability using the Vision Transformer model.

Listing A7. Getting predictions for images from external source.

```
labels = ["airport", "bare-land", "beach", "bridge", "commercial", "desert", "
    farmland", "forest", "golf-course", "highway", "industrial", "meadow", "
    mountain", "overpass", "park", "parking", "playground", "port", "railway", "
    railway-station", "residential", "river", "runway", "stadium", "storage-tank"
    ]
transform = ResizeCenterCropToTensor()
image = image_loader('/images/image1.tif')
fig = model.predict_image(image, labels, transform)
```

Appendix D. Semantic Segmentation of Remote Sensing Images

A Jupyter Notebook for demonstrating the task of semantic segmentation of remote sensing images is available in the AiTLAS repository (https://github.com/biasvariancelabs/aitlas/blob/master/examples/semantic_segmentation_example_landcover_ai.ipynb (accessed on 8 March 2023)). The notebook provides the code needed for loading a semantic segmentation dataset and examining the distribution of the pixels across the semantic labels/classes. The notebook also provides instructions on how to train and evaluate the DeepLabv3 model and how to use the learned model for running predictions on unseen data.

In this example, we use the LandCover.ai image classification dataset. To obtain the dataset, we download it from the repository (https://landcover.ai.linuxpolska.com/download/landcover.ai.v1.zip (accessed on 8 March 2023)) and unzip it, after which we obtain folders with images and masks, as well as separate files containing information about the train, validation, and test splits.

Next, we create an instance of the class *LandCoverAiDataset* available in the toolbox. To create the instance, we provide the folder with the images and masks and a file that contains the list of images to be loaded. The class *LandCoverAiDataset* has all the functionalities to load the images and masks for processing and training. Additionally, in the data configuration object, we set the batch size, the shuffle parameter, and the number of workers for loading the data (see Listing A8).

Listing A8. Load the LandCoverAiDataset dataset from the AiTLAS toolbox, inspect images and masks, and calculate the pixel distribution across the labels.

```
dataset_config = {
"data_dir": "/dataset/landcoverai/images",
"csv_file": "/dataset/landcoverai/train.txt",
"batch_size": 128,
"shuffle": True,
"num_workers": 4
}
dataset = LandCoverAiDataset(dataset_config)
dataset.show_image(2000);
dataset.data_distribution_table()
dataset.data_distribution_barchart();
```

We use the `show_image` function to display images from the dataset. In the displayed image, the different semantic regions in the mask are color coded with the colors from the class definition of the dataset. To inspect the distribution of the pixels across the labels in the dataset, we use the `data_distribution_table` and/or `data_distribution_barchart` function.

We continue with the model learning task by creating the training and validation datasets and training the deep-learning model. The code snippet for creating train and validation datasets is given in Listing A9. Additionally, Listing A9 shows an example of specifying the data transformation parameter. In the AiTLAS toolbox, three different transformations are available: transformation on the images, transformations on the targets (i.e., labels, masks, or bounding boxes), and joint transformations that are simultaneously applied on the input images and the targets. For example, in the case of semantic segmentation, if the input image is horizontally flipped, the mask should also be flipped. During training, in this use case, we perform *data augmentation* by random horizontal and/or vertical flips on the input images and the masks. For the input images, min-max normalization is applied, and the targets/masks are transposed. During evaluation/testing, we do not use joint transformations.

Listing A9. Load the training and validation dataset for semantic segmentation.

```
train_dataset_config = {
"batch_size": 16,
"shuffle": True,
"csv_file": "/dataset/landcoverai/train.txt",
"data_dir": "/dataset/landcoverai/images",
"joint_transforms": ["aitlas.transforms.FlipHVRandomRotate"],
"transforms": ["aitlas.transforms.MinMaxNormTranspose"],
"target_transforms": ["aitlas.transforms.Transpose"]
}
train_dataset = LandCoverAiDataset(train_dataset_config)

validation_dataset_config = {
"batch_size": 16,
"shuffle": False,
"csv_file": "../dataset/landcoverai/val.txt",
"data_dir": "../dataset/landcoverai/images",
"transforms": ["aitlas.transforms.MinMaxNormTranspose"],
"target_transforms": ["aitlas.transforms.Transpose"]
}
validation_dataset = LandCoverAiDataset(validation_dataset_config)
```

We use the DeepLabv3 model, which is available in the AiTLAS toolbox. We configure the model by setting several configuration parameters, i.e., the number of classes/labels, the learning rate, the pretraining mode, and the evaluation metrics. The code snippet for instantiating the model and running the training sequence is given in Listing A10.

Listing A10. Creating a DeepLabv3 model and starting the training.

```
epochs = 100
model_directory = "/experiments/landcoverai"
model_config = {
"num_classes": 5,
"learning_rate": 0.0001,
"pretrained": True,
"threshold": 0.5,
"metrics": ["iou"]
}

model = DeepLabV3(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
val_dataset=validation_dataset,
epochs=epochs,
model_directory=model_directory,
run_id='1'
)
```

We train two variants of DeepLabV3: (i) model "trained from scratch" using only the dataset at hand and initialized with random weights at the start of the training procedure, and (2) model with pre-trained weights on a subset of the COCO dataset, using only the 20 categories that are also present in the Pascal VOC dataset [66] and then fine-tuned on the dataset at hand. The DeepLabv3 model is trained or fine-tuned on the train set of images, including *data augmentation* operations such as random horizontal and/or vertical flips and random rotations. Using the validation set of images, we perform a hyper-parameter search over different values for the learning rate: 0.01, 0.001, and 0.0001. We use fixed values for some of the hyper-parameters: batch size is set to 16, *Adam optimizer* [146] without weight decay, and *ReduceLROnPlateau* as a learning scheduler which reduces the learning rate when the loss has stopped improving (same as for the image scene classification task as discussed in Section 3.1.2). Furthermore, to prevent over-fitting, we perform early stopping on the validation set—the model with the lowest validation loss is saved and then applied on the original test set to obtain the estimate of the model's predictive performance. Finally, we use mean intersection over union ($mIoU$) as an evaluation measure: $mIoU$ is a widely

used measure for semantic segmentation and is calculated as the average of intersection over union (IoU) across all labels. Note that IoU is defined as the area of the overlap between the ground truth and predicted label divided by the area of their union (a value of 0 for IoU means no overlap, while a value of 1 means complete overlap).

The train and validation learning curves are shown in Figure A3. After the 50 epoch, the value of the loss function is very stable, and the model starts to over-fit. The stop criteria prevent over-fitting. At this point, we save the model from the 49th epoch (with the lowest validation loss) and evaluate its performance on the test set.

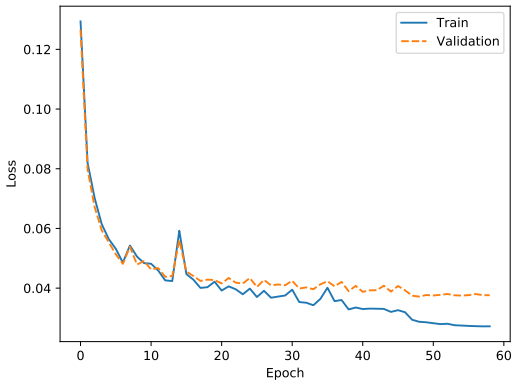


Figure A3. Train and validation learning curves for the DeepLabv3 model pre-trained on COCO and fine-tuned on the LandCover.ai dataset.

To evaluate the model, we load the test split of the dataset, set the path to the learned model, and set the list of evaluation metrics. In this example, we use IoU as an evaluation measure. We then proceed with running the evaluation sequence on the test data (see Listing A11).

Listing A11. Testing the model using the images from the test split.

```
test_dataset_config = {
"batch_size": 4,
"shuffle": False,
"data_dir": "/dataset/landcoverai/images",
"csv_file": "/dataset/landcoverai/test.txt",
"transforms": ["aitlas.transforms.MinMaxNormTranspose"],
"target_transforms": ["aitlas.transforms.Transpose"]
}

test_dataset = LandCoverAiDataset(test_dataset_config)
model_path = "/experiments/landcoverai/best_checkpoint.pth.tar"
model.metrics = ["iou"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

The results summarizing the segmentation performance of the DeepLabV3 model on the LandCover.ai dataset are given in Table A4. The DeepLabv3 model trained from scratch yields a $mIoU$ score of 86.027%, while the pre-trained one yields a score of 86.093%. Hence, there is no clear benefit of using the pre-trained weights for this particular dataset because the results in the different setups are very similar. The values of the IoU for the labels 'Road' and 'Building' are lower compared to the other labels. These labels are more challenging in the context of semantic segmentation because they are usually narrow, in the case of roads, or often small, in the case of buildings. Additionally, they are sometimes obscured by other objects, such as trees.

Table A4. Label-wise $IoU(\%)$ and $mIoU$ for the DeepLabv3 models trained on the LandCover.ai dataset.

Model/Label	Background	Buildings	Woodlands	Water	Road	mIoU	Avg. Time/Epoch (s)	Total Time (s)	Best Epoch
Trained from scratch	93.813	80.304	91.952	94.877	69.190	86.027	299.241	16159	44
Pre-trained on COCO	93.857	80.650	91.964	95.145	68.846	86.093	300.58	17734	49

Additionally, the learned model can be used for predicting the segmentation masks of unseen images by using the `predict_masks` function. The function takes the image, the labels, and the transformation instance as arguments and returns separate masks for each semantic label for the given image. An example output of this function is shown in Figure A4. The code snippet to obtain the prediction for a given image is shown in Listing A12.

Listing A12. Getting predictions for images from external source.

```
labels = ["Background", "Buildings", "Woodlands", "Water", "Road"]
transform = MinMaxNormTranspose()
model_path = "/experiments/landcoverai/best_checkpoint.pth.tar"
model.load_model(model_path)
image = image_loader('/images/image1.png')
fig = model.predict_masks(image, labels, transform)
```

Figure A4. Example image with the predicted masks for each semantic label using the DeepLabv3 model.

Appendix E. Remote Sensing Image Object Detection

An example Jupyter Notebook for object detection in remote sensing images is available at the AiTLAS toolbox (https://github.com/biasvariancelabs/aitlas/blob/master/examples/object_detection_example_hrrsd.ipynb (accessed on 8 March 2023)). The notebook provides step-by-step code on how to load an object detection dataset and inspect the number of object instances for each object category. The notebook also demonstrates how to train and validate the Faster R-CNN model on the predefined test split, as well as how to use the learned model for inference on unseen data.

To prepare the HRRSD object detection dataset, we need to download the data from the repository (https://github.com/CrazyStoneonRoad/TGRS-HRRSD-Dataset (accessed on 8 March 2023)) and unzip it. This generates a folder with images in .jpg format, annotations in .xml format, and separate files containing information about the train, validation, and test splits. The Pascal VOC object annotation format is used in this example. Thus, we create an instance of the *ObjectDetectionPascalDataset* class defined within the AiTLAS toolbox. To instantiate the class, we specify the path to the folders containing the images and annotations and the file containing the list of images. We also set the batch size for the data loader during training, the shuffle parameter, and the number of workers to specify the number of sub-processes used for data loading (see Listing A13).

Listing A13. Load a dataset using the AiTLAS toolbox, inspect images and calculate the number of object instances for each category/label.

```
dataset_config = {
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/train.txt"
"batch_size": 16,
"shuffle": True,
"num_workers": 4
}
dataset = ObjectDetectionPascalDataset(dataset_config)
dataset.show_image(2458);
dataset.show_batch(15);
dataset.data_distribution_table()
dataset.data_distribution_barchart();
```

We use the `show_image` function to display images from the dataset. The function displays the image with the bounding boxes and labels for each object present in the image. The function `show_batch` displays a batch of images with the bounding boxes of the objects in the image. The statistics for the number of object instances for each category/label in the dataset can be calculated using the functions `data_distribution_table` or `data_distribution_barchart`.

In the next step, we continue with the model learning part. First, we create the configuration objects for the training and validation data and specify the data transformation method we wish to be applied. For this use case, we apply the joint transformation *ResizeToTensorV2* to resize the images to a resolution of 480×480 pixels and convert them to tensors. This transformation also resizes the bounding boxes for the object to fit the new resolution of the images. After defining the data configuration objects, we load the data (see Listing A14).

Listing A14. Load the training and validation dataset.

```
train_dataset_config = {
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/train.txt",
"joint_transforms": ["aitlas.transforms.ResizeToTensorV2"]
"batch_size": 16,
"shuffle": True,
"num_workers": 4
}
train_dataset = ObjectDetectionPascalDataset(train_dataset_config)

validation_dataset_config = {
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/val.txt",
"joint_transforms": ["aitlas.transforms.ResizeToTensorV2"]
"batch_size": 16,
"shuffle": True,
"num_workers": 4
}
validation_dataset = ObjectDetectionPascalDataset(validation_dataset_config)
```

We train two variants of the model: (i) model "trained from scratch" using only the dataset at hand and initialized with random weights at the start of the training procedure, and (2) model with pre-trained weights on the COCO dataset [66] and then fine-tuned on the dataset at hand. The Faster R-CNN model is trained or fine-tuned using the training part of the images, with parameters selection/search performed using the validation part. Namely, we search over different values for the learning rate: 0.01, 0.001, and 0.0001. We use fixed values for some of the hyper-parameters: batch size is set to 16, *Adam optimizer* [146] without weight decay, and *ReduceLROnPlateau* as a learning scheduler which reduces the learning rate when the loss has stopped improving.

Furthermore, to prevent over-fitting, we perform early stopping on the validation set—the model with the best value for the evaluation measure is saved and then applied to the original test set to obtain the estimate of the model's predictive performance. Finally, as an evaluation measure, we use mean Average Precision (mAP) as defined in the Pascal VOC Challenge [121]. Average Precision (AP) is computed as the average precision value taken at recall values ranging from 0 to 1 (i.e., the area under the precision/recall curve). mAP is the average of AP over all classes. Next, IoU is crucial in determining true positives and false positives, and its threshold is set to 0.5. More details on the evaluation measures for object detection are provided in [121,142].

Next, we use the Faster R-CNN model, which is implemented in the AiTLAS toolbox. To configure the model, create a model configuration object and set several configuration parameters, i.e., the number of classes/labels, the learning rate, the pretraining mode, and the evaluation metrics. The code snippet for instantiating the model and running the training sequence is given in Listing A15.

Listing A15. Creating a Faster R-CNN model and start of model training.

```
epochs = 100
model_directory = "/experiments/hrrsd"
model_config = {
"num_classes": 14,
"learning_rate": 0.0001,
"pretrained": True,
"threshold": 0.5,
"metrics": ["map"]
}
model = FasterRCNN(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
val_dataset=validation_dataset,
epochs=epochs,
model_directory=model_directory,
run_id='1'
)
```

Table A5 summarizes the results of the object detection task. The Faster R-CNN model trained from scratch yields 77.412% of mAP, while the pre-trained model yields 81.436% of mAP, as estimated using the test dataset. The results show the clear benefit of using the pre-trained weights for this particular dataset. The most challenging objects to detect are 'Crossroad' and 'T Junction' (due to the similarity of both objects).

Table A5. Mean average precision (mAP%) of the Faster R-CNN models trained from scratch and pre-trained for the HRRSD dataset.

Label	Faster R-CNN (Pretrained)	Faster R-CNN
Airplane	96.86	94.71
Baseball Diamond	79.75	80.13
Basketball Court	59.25	44.96
Bridge	82.22	78.55
Crossroad	77.06	71.78
Ground Track Field	95.62	92.84
Harbor	89.00	88.61
Parking Lot	53.80	51.22
Ship	86.61	78.50
Storage Tank	93.56	89.67
T Junction	66.83	69.12
Tennis Court	87.97	79.31
Vehicle	90.14	86.96
Mean AP	81.436	77.412
Avg. time / epoch (s)	221.96	244.63
Total time (s)	5993	10030
Best epoch	17	31

To evaluate the model, we create a configuration object for the training split of the HRRSD data, load the data, set the path to the trained model, and run the evaluation process (see Listing A16).

Listing A16. Testing the model using the images from the test split.

```
test_dataset_config = {
"batch_size": 4,
"shuffle": False,
"image_dir": "/datasets/HRRSD/images",
"annotations_dir": "/datasets/HRRSD/annotations",
"imageset_file": "/datasets/HRRSD/test.txt",
"joint_transforms": ["aitlas.transforms.ResizeToTensorV2"]
}

test_dataset = ObjectDetectionPascalDataset(test_dataset_config)
model_path = "/experiments/hrrsd/best_checkpoint.pth.tar"
model.metrics = ["map"]
model.running_metrics.reset()
model.evaluate(dataset=test_dataset, model_path=model_path)
model.running_metrics.get_scores(model.metrics)
```

Additionally, the model can be used for predicting the bounding boxes and labels for the objects in new, unseen images from an external source by using the `predict_objects` function. The function takes the image, the labels, and the transformation instance as arguments and returns an image with the bounding boxes and labels for the detected object in the image. Figure A5 shows an example output of this function. The code snippet to obtain the prediction for a given image is shown in Listing A17.

Listing A17. Getting bounding boxes and labels for images from external source.

```
labels = [None, 'T junction', 'airplane', 'baseball diamond', 'basketball court',
'bridge', 'crossroad', 'ground track field', 'harbor', 'parking lot',
'ship', 'storage tank', 'tennis court', 'vehicle']
transform = Resize()
model.load_model(model_path)
image = image_loader('../data/HRRSD/JPEGImages/00042.jpg')
fig = model.detect_objects(image, labels, transform)
```

Figure A5. Example image with the predicted bounding boxes and labels for the objects using the Faster R-CNN model.

Appendix F. Crop Type Prediction Using Satellite Time Series Data

We have added a Jupyter Notebook (https://github.com/biasvariancelabs/aitlas/blob/master/examples/crop_type_prediction_example_netherlands.ipynb (accessed on 8

March 2023)) in the AiTLAS toolbox that contains a step-by-step sample code for running through a crop type prediction task. As previously, we will demonstrate the standard steps to load and inspect the data, load, configure, train, and evaluate the models, and visualize predictions.

We use the AiTLAS NLD dataset for the year 2019. First, we set the configuration for the dataset and initiate with the `EOPatchCrops` class, which is a wrapper for working with `EOPatches`. We need to set the path to the root folder containing the patches. The patches are also folders containing detailed EO data. We need to specify the index, which is a `csv` file containing the class mappings for the polygons and patches they belong to, as well as the split (train, validation, or test). We also specify the standard PyTorch attributes for working with datasets, such as batch size, shuffle, and number of workers. The `regions` configurations specify which splits we should load from the index. Sample code is shown in Listing A18.

Listing A18. Load the dataset.

```
dataset_config = {
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"batch_size": 128,
"shuffle": True,
"num_workers": 4,
"regions":["train", "test", "val",],
}
dataset = EOPatchCrops(dataset_config)
```

To display the time series values of the bands of a specific polygon, we can use the function `show_timeseries(index)`. Furthermore, to show sample data from the underlying index, we can use the `show_samples()` function (Listing A19).

Listing A19. Inspect the dataset.

```
fig = dataset.show_timeseries(0)
dataset.show_samples()
```

An example of the data representation for a selected polygon labeled as permanent grassland is illustrated in Figure A6.

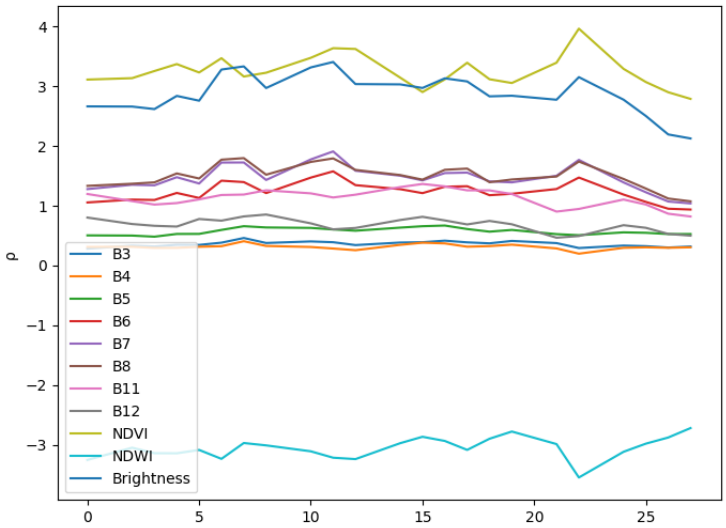


Figure A6. Examples of the input time series of reflectances ρ for the spectral bands of the Sentinel 2 satellite and calculated indices for the crop type *Permanent grassland*.

Once, we have a sense of the data, we load the train and validation splits of the dataset (Listing A20).

Listing A20. Load train and validation the dataset.

```
train_dataset_config = {
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"batch_size": 128,
"shuffle": True,
"num_workers": 4,
"regions":["train", ],
}

train_dataset = EOPatchCrops(train_dataset_config)

validation_dataset_config = {
"batch_size": 32,
"shuffle": False,
"num_workers": 4,
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"regions":["val", ],
}

validation_dataset = EOPatchCrops(validation_dataset_config)
```

In the next step, we need to initialize and create the model. For this task, we will use the LSTM model supported by the toolbox. We set the model parameters in the configuration object, instantiate the model, and run the training sequence (Listing A21).

Listing A21. Train the model.

```
epochs = 100
model_directory = "./experiments/LSTM"
model_config = {
"input_dim":11,
"num_classes": 10,
"learning_rate": 0.001,
"dropout" : 0.2,
"weight_decay": 0.0001,
"metrics":["accuracy","f1_score", "kappa"]
}
model = LSTM(model_config)
model.prepare()
model.train_and_evaluate_model(
train_dataset=train_dataset,
epochs=epochs,
model_directory=model_directory,
val_dataset=validation_dataset,
run_id='1',
)
```

We can use the model to run predictions and visualize them. We can load the test split of the dataset. Then, use the model to run predictions on that dataset (Listing A22). Finally, we can pick a sample patch and visualize its predicted polygons (Figure A7).

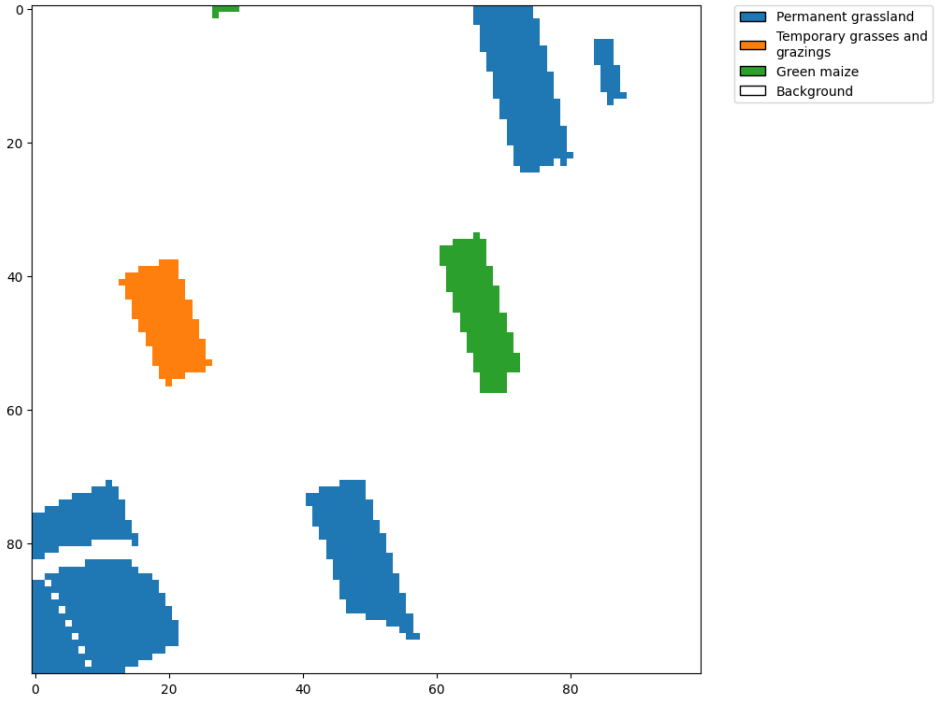


Figure A7. Example patch with the predicted polygons using the LSTM model.

Listing A22. Test the model.

```
labels = ["Permanent grassland", "Temporary grasses and grazings", "Green maize",
    "Potatoes (including seed potatoes)",
"Common winter wheat and spelt", "Sugar beet (excluding seed)", "Other farmland",
    "Onions",
"Flowers and ornamental plants (excluding nurseries)", "Spring barley",]

test_dataset_config = {
"batch_size": 32,
"shuffle": False,
"num_workers": 4,
"root": "/home/user/data/CropTypeNetherlands/2019/",
"csv_file_path": "index.csv",
"regions":["test", ],
}

test_dataset = EOPatchCrops(test_dataset_config)

y_true, y_pred, y_prob = model.predict(dataset=test_dataset,)

eopatches_path = "/home/user/data/CropTypeNetherlands/2019/eopatches/"
patch = "eopatch_7495"

fig = display_eopatch_predictions(
eopatches_path,
patch,
y_pred,
test_dataset.index,
y_true,
test_dataset.mapping,
)
```

The results summarizing the predictive performance on the AiTLAS NLD dataset across the different years are presented in Table A6. Using LSTM, we obtain a weighted F1 score in the range of ∼82–84% for the different years.

Table A6. Results for the crop type prediction on the AiTLAS NLD dataset using the LSTM model.

Dataset Year	Accuracy	Weighted F1 Score	Kappa
2017	84.28	82.48	77.22
2018	84.49	84.32	78.79
2019	85.25	84.10	79.55

We also present the per class F1 scores across all years in Table A7. We can note that the per-class performance is generally consistent between different years. The F1 score does not change significantly for any class across the different years. The only exception is the *Temporary grasses and grazings* class, where for 2017, the F1 score is 45.23, while for 2018, it is 59.34, which is around a 14% difference. The performance for this class is also the lowest compared to the other classes. The model performs best on the classes *Green maize*, *Common winter wheat and spelt*, *Potatoes (including seed potatoes)*, and *Sugar beet (excluding seed)*. For these classes, the F1 score is consistent across the different years with score ∼95%. We provide the resources and explanations on the execution of this use case, including additional visualizations and application of the model to external images in Appendix F.

Table A7. Per class F1 score for the crop type prediction on the AiTLAS NLD dataset for every year with LSTM

Crop Type	2017	2018	2019
Permanent grassland	86.72	85.78	86.82
Temporary grasses and grazings	45.23	59.34	53.66
Green maize	96.10	95.09	96.14
Potatoes (including seed potatoes)	95.37	94.84	95.45
Common winter wheat and spelt	94.86	95.75	96.28
Sugar beet (excluding seed)	95.62	92.36	95.28
Other farmland	62.45	61.81	60.05
Onions	89.76	93.68	92.51
Flowers and ornamental plants (excluding nurseries)	83.18	78.30	79.67
Spring barley	92.30	91.12	90.80

References

1. Christopherson, J.; Chandra, S.N.R.; Quanbeck, J.Q. *2019 Joint Agency Commercial Imagery Evaluation—Land Remote Sensing Satellite Compendium*; Technical Report; US Geological Survey: Reston, VA, USA, 2019.
2. Tupin, F.; Inglada, J.; Nicolas, J.M. *Remote Sensing Imagery*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
3. Dimitrovski, I.; Kitanovski, I.; Kocev, D.; Simidjievski, N. Current trends in deep learning for Earth Observation: An open-source benchmark arena for image classification. *ISPRS J. Photogramm. Remote Sens.* **2023**, *197*, 18–35. [CrossRef]
4. Cheng, G.; Han, J.; Lu, X. Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proc. IEEE* **2017**, *105*, 1865–1883. [CrossRef]
5. Tang, L.; Shao, G. Drone remote sensing for forestry research and practices. *J. For. Res.* **2015**, *26*, 791–797. [CrossRef]
6. Roy, P.; Ranganath, B.; Diwakar, P.; Vohra, T.; Bhan, S.; Singh, I.; Pandian, V. Tropical forest typo mapping and monitoring using remote sensing. *Remote Sens.* **1991**, *12*, 2205–2225. [CrossRef]
7. Sunar, F.; Özkan, C. Forest fire analysis with remote sensing data. *Int. J. Remote Sens.* **2001**, *22*, 2265–2277. [CrossRef]
8. Poursanidis, D.; Chrysoulakis, N. Remote Sensing, natural hazards and the contribution of ESA Sentinels missions. *Remote Sens. Appl. Soc. Environ.* **2017**, *6*, 25–38. [CrossRef]
9. Sishodia, R.P.; Ray, R.L.; Singh, S.K. Applications of remote sensing in precision agriculture: A review. *Remote Sens.* **2020**, *12*, 3136. [CrossRef]
10. Cox, H.; Kelly, K.; Yetter, L. Using remote sensing and geospatial technology for climate change education. *J. Geosci. Educ.* **2014**, *62*, 609–620. [CrossRef]
11. Collis, R.T.; Creasey, D.; Grasty, R.; Hartl, P.; deLoor, G.; Russel, P.; Salerno, A.; Schaper, P. *Remote Sensing for Environmental Sciences*; Springer Science & Business Media: Berlin, Germany, 2012; Volume 18.
12. Christie, G.; Fendley, N.; Wilson, J.; Mukherjee, R. Functional Map of the World. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6172–6180. [CrossRef]

13. Sumbul, G.; de Wall, A.; Kreuziger, T.; Marcelino, F.; Costa, H.; Benevides, P.; Caetano, M.; Demir, B.; Markl, V. BigEarthNet-MM: A Large-Scale, Multimodal, Multilabel Benchmark Archive for Remote Sensing Image Classification and Retrieval [Software and Data Sets]. *IEEE Geosci. Remote Sens. Mag.* **2021**, *9*, 174–180. [CrossRef]
14. Long, Y.; Xia, G.S.; Li, S.; Yang, W.; Yang, M.Y.; Zhu, X.X.; Zhang, L.; Li, D. On Creating Benchmark Dataset for Aerial Image Interpretation: Reviews, Guidances and Million-AID. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 4205–4230. [CrossRef]
15. Bastani, F.; Wolters, P.; Gupta, R.; Ferdinando, J.; Kembhavi, A. Satlas: A Large-Scale, Multi-Task Dataset for Remote Sensing Image Understanding. *arXiv* **2022**, arXiv:2211.15660. https://doi.org/10.48550/ARXIV.2211.15660.
16. Neupane, B.; Horanont, T.; Aryal, J. Deep Learning-Based Semantic Segmentation of Urban Features in Satellite Images: A Review and Meta-Analysis. *Remote Sens.* **2021**, *13*, 808. [CrossRef]
17. Xia, G.S.; Bai, X.; Ding, J.; Zhu, Z.; Belongie, S.; Luo, J.; Datcu, M.; Pelillo, M.; Zhang, L. DOTA: A Large-Scale Dataset for Object Detection in Aerial Images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
18. Pearlman, J.; Barry, P.; Segal, C.; Shepanski, J.; Beiso, D.; Carman, S. Hyperion, a space-based imaging spectrometer. *IEEE Trans. Geosci. Remote Sens.* **2003**, *41*, 1160–1173. [CrossRef]
19. King, M.; Herring, D. SATELLITES | Research (Atmospheric Science). In *Encyclopedia of Atmospheric Sciences*; Holton, J.R., Ed.; Academic Press: Oxford, UK, 2003; pp. 2038–2047. . [CrossRef]
20. Osco, L.P.; Marcato Junior, J.; Marques Ramos, A.P.; de Castro Jorge, L.A.; Fatholahi, S.N.; de Andrade Silva, J.; Matsubara, E.T.; Pistori, H.; Gonçalves, W.N.; Li, J. A review on deep learning in UAV remote sensing. *Int. J. Appl. Earth Obs. Geoinf.* **2021**, *102*, 102456. [CrossRef]
21. Roy, D.; Wulder, M.; Loveland, T.; C.E., W.; Allen, R.; Anderson, M.; Helder, D.; Irons, J.; Johnson, D.; Kennedy, R.; et al. Landsat-8: Science and product vision for terrestrial global change research. *Remote Sens. Environ.* **2014**, *145*, 154–172. [CrossRef]
22. Drusch, M.; Del Bello, U.; Carlier, S.; Colin, O.; Fernandez, V.; Gascon, F.; Hoersch, B.; Isola, C.; Laberinti, P.; Martimort, P.; et al. Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services. *Remote Sens. Environ.* **2012**, *120*, 25–36.
23. Sentinel Hub: Cloud API For Satellite Imagery. 2023. Available online: https://www.sentinel-hub.com/ (accessed on 8 March 2023).
24. UP42: Simplified Access to Geospatial Data and Processing. 2023. Available online: https://up42.com/ (accessed on 8 March 2023).
25. De Vroey, M.; Radoux, J.; Zavagli, M.; De Vendictis, L.; Heymans, D.; Bontemps, S.; Defourny, P. Performance Assessment of the Sen4CAP Mowing Detection Algorithm on a Large Reference Data Set of Managed Grasslands. In Proceedings of the 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS, Brussels, Belgium, 11–16 July 2021; pp. 743–746. [CrossRef]
26. Grizonnet, M.; Michel, J.; Poughon, V.; Inglada, J.; Savinaud, M.; Cresson, R. Orfeo ToolBox: open source processing of remote sensing images. *Open Geospat. Data Softw. Stand.* **2017**, *2*, 15. [CrossRef]
27. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *arXiv* **2015**, arXiv:1603.04467.
28. Rolland, J.F.o.; Castel, F.; Haugommard, A.; Aubrun, M.; Yao, W.; Corneliu.; Dumitru, O.; Datcu, M.; Bylicki, M.; Tran, B.H.; et al. Candela: A Cloud Platform for Copernicus Earth Observation Data Analytics. In Proceedings of the IGARSS 2020—2020 IEEE International Geoscience and Remote Sensing Symposium, Waikoloa, HI, USA, 26 September–2 October 2020; pp. 3104–3107. [CrossRef]
29. Stewart, A.J.; Robinson, C.; Corley, I.A.; Ortiz, A.; Lavista Ferres, J.M.; Banerjee, A. TorchGeo: Deep Learning with Geospatial Data. In Proceedings of the SIGSPATIAL '22: 30th International Conference on Advances in Geographic Information Systems; Association for Computing Machinery: Seattle, WA, USA, 2022; pp. 1–12. [CrossRef]
30. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
31. Chaudhuri, B.; Demir, B.; Chaudhuri, S.; Bruzzone, L. Multilabel Remote Sensing Image Retrieval Using a Semisupervised Graph-Theoretic Method. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 1144–1158. [CrossRef]
32. Tsoumakas, G.; Katakis, I. Multi-Label Classification: An Overview. *Int. J. Data Warehous. Min.* **2009**, *3*, 1–13. [CrossRef]
33. Yang, Y.; Newsam, S. Bag-of-Visual-Words and Spatial Extensions for Land-Use Classification. In Proceedings of the GIS '10: 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, CA, USA, 2–5 November 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 270–279.
34. Xia, G.S.; Yang, W.; Delon, J.; Gousseau, Y.; Sun, H.; Maître, H. Structural High-resolution Satellite Image Indexing. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. ISPRS Arch.* **2010**, *38*, 1–6.
35. Xia, G.S.; Hu, J.; Hu, F.; Shi, B.; Bai, X.; Zhong, Y.; Zhang, L.; Lu, X. AID: A Benchmark Data Set for Performance Evaluation of Aerial Scene Classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 3965–3981. [CrossRef]
36. Helber, P.; Bischke, B.; Dengel, A.; Borth, D. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2019**, *12*, 2217–2226. [CrossRef]
37. Zhou, W.; Newsam, S.; Li, C.; Shao, Z. PatternNet: A benchmark dataset for performance evaluation of remote sensing image retrieval. *ISPRS J. Photogramm. Remote Sens.* **2018**, *145*, 197–209. [CrossRef]
38. Li, H.; Dou, X.; Tao, C.; Wu, Z.; Chen, J.; Peng, J.; Deng, M.; Zhao, L. RSI-CB: A Large-Scale Remote Sensing Image Classification Benchmark Using Crowdsourced Data. *Sensors* **2020**, *20*, 1594. [CrossRef] [PubMed]
39. Zou, Q.; Ni, L.; Zhang, T.; Wang, Q. Deep Learning Based Feature Selection for Remote Sensing Scene Classification. *IEEE Geosci. Remote Sens. Lett.* **2015**, *12*, 2321–2325. [CrossRef]

40. Basu, S.; Ganguly, S.; Mukhopadhyay, S.; DiBiano, R.; Karki, M.; Nemani, R. DeepSat: A Learning Framework for Satellite Imagery. In *SIGSPATIAL '15: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*; Association for Computing Machinery: New York, NY, USA, 2015.
41. Zhu, Q.; Zhong, Y.; Zhao, B.; Xia, G.S.; Zhang, L. Bag-of-Visual-Words Scene Classifier with Local and Global Features for High Spatial Resolution Remote Sensing Imagery. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 747–751. [CrossRef]
42. Li, H.; Jiang, H.; Gu, X.; Peng, J.; Li, W.; Hong, L.; Tao, C. CLRS: Continual Learning Benchmark for Remote Sensing Image Scene Classification. *Sensors* **2020**, *20*, 1226. [CrossRef]
43. Long, Y.; Gong, Y.; Xiao, Z.; Liu, Q. Accurate Object Localization in Remote Sensing Images Based on Convolutional Neural Networks. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 2486–2498. [CrossRef]
44. Wang, Q.; Liu, S.; Chanussot, J.; Li, X. Scene Classification With Recurrent Attention of VHR Remote Sensing Images. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 1155–1167. [CrossRef]
45. Penatti, O.A.; Nogueira, K.; Dos Santos, J.A. Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Boston, MA, USA, 7–12 June 2015; pp. 44–51.
46. Zhu, X.X.; Hu, J.; Qiu, C.; Shi, Y.; Kang, J.; Mou, L.; Bagheri, H.; Haberle, M.; Hua, Y.; Huang, R.; et al. So2Sat LCZ42: A Benchmark Data Set for the Classification of Global Local Climate Zones [Software and Data Sets]. *IEEE Geosci. Remote Sens. Mag.* **2020**, *8*, 76–89. [CrossRef]
47. Qi, X.; Zhu, P.; Wang, Y.; Zhang, L.; Peng, J.; Wu, M.; Chen, J.; Zhao, X.; Zang, N.; Mathiopoulos, P.T. MLRSNet: A multi-label high spatial resolution remote sensing dataset for semantic scene understanding. *ISPRS J. Photogramm. Remote Sens.* **2020**, *169*, 337–350. [CrossRef]
48. Hua, Y.; Mou, L.; Zhu, X.X. Recurrently exploring class-wise attention in a hybrid convolutional and bidirectional LSTM network for multi-label aerial image classification. *ISPRS J. Photogramm. Remote Sens.* **2019**, *149*, 188–199. [CrossRef]
49. Sumbul, G.; Charfuelan, M.; Demir, B.; Markl, V. Bigearthnet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding. In Proceedings of the IGARSS 2019—2019 IEEE International Geoscience and Remote Sensing Symposium Yokohama, Japan, 28 July–2 August 2019; pp. 5901–5904.
50. Hua, Y.; Mou, L.; Zhu, X.X. Relation Network for Multilabel Aerial Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 4558–4572. [CrossRef]
51. Kaggle. Planet: Understanding the Amazon from Space. 2022. Available online: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space (accessed on 8 March 2023).
52. Zhang, Y.; Yuan, Y.; Feng, Y.; Lu, X. Hierarchical and Robust Convolutional Neural Network for Very High-Resolution Remote Sensing Object Detection. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 5535–5548. [CrossRef]
53. Ding, J.; Xue, N.; Xia, G.S.; Bai, X.; Yang, W.; Yang, M.Y.; Belongie, S.; Luo, J.; Datcu, M.; Pelillo, M.; et al. Object Detection in Aerial Images: A Large-Scale Benchmark and Challenges. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 7778–7796. [CrossRef]
54. Li, K.; Wan, G.; Cheng, G.; Meng, L.; Han, J. Object detection in optical remote sensing images: A survey and a new benchmark. *ISPRS J. Photogramm. Remote Sens.* **2020**, *159*, 296–307. . [CrossRef]
55. Cheng, G.; Han, J.; Zhou, P.; Guo, L. Multi-class geospatial object detection and geographic image classification based on collection of part detectors. *ISPRS J. Photogramm. Remote Sens.* **2014**, *98*, 119–132. [CrossRef]
56. Haroon, M.; Shahzad, M.; Fraz, M.M. Multisized object detection using spaceborne optical imagery. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2020**, *13*, 3032–3046. [CrossRef]
57. Mnih, V. Machine Learning for Aerial Image Labeling. Ph.D. Thesis, University of Toronto, Toronto, ON, Canada, 2013.
58. Boguszewski, A.; Batorski, D.; Ziemba-Jankowska, N.; Dziedzic, T.; Zambrzycka, A. LandCover.ai: Dataset for Automatic Mapping of Buildings, Woodlands, Water and Roads from Aerial Imagery. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Nashville, TN, USA, 19–25 June 2021; pp. 1102–1110.
59. Maggiori, E.; Tarabalka, Y.; Charpiat, G.; Alliez, P. Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark. In Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Fort Worth, TX, USA, 23–28 July 2017.
60. Chen, Q.; Wang, L.; Wu, Y.; Wu, G.; Guo, Z.; Waslander, S.L. Aerial imagery for roof segmentation: A large-scale dataset towards automatic mapping of buildings. *ISPRS J. Photogramm. Remote Sens.* **2019**, *147*, 42–55. [CrossRef]
61. Bragagnolo, L.; da Silva, R.V.; Grzybowski, J.M.V. Amazon Rainforest Dataset for Semantic Segmentation. 2019. Available online: https://zenodo.org/record/3233081#.ZENXKc5ByUk (accessed on 8 March 2023).
62. Kocev, D.; Simidjievski, N.; Kostovska, A.; Dimitrovski, I.; Kokalj, Z. Discover the Mysteries of the Maya: Selected Contributions from the Machine Learning Challenge: The Discovery Challenge Workshop at ECML PKDD 2021. *arXiv* **2022**, arXiv:2208.03163. https://doi.org/10.48550/arXiv.2208.03163.
63. Merdjanovska, E.; Kitanovski, I.; Kokalj, Ž.; Dimitrovski, I.; Kocev, D. Crop Type Prediction Across Countries and Years: Slovenia, Denmark and the Netherlands. In Proceedings of the IGARSS 2022—2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur, Malaysia, 17–22 July 2022; pp. 5945–5948.
64. Rußwurm, M.; Lefèvre, S.; Körner, M. Breizhcrops: A satellite time series dataset for crop type identification. In Proceedings of the International Conference on Machine Learning Time Series Workshop, Long Beach, CA, USA, 9–15 June 2019; Volume 3.

65. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
66. Marcel, S.; Rodriguez, Y. Torchvision the machine-vision package of torch. In Proceedings of the 18th ACM International Conference on Multimedia, Firenze, Italy, 25–29 October 2010; pp. 1485–1488.
67. Wang, J.; Yang, Y.; Mao, J.; Huang, Z.; Huang, C.; Xu, W. CNN-RNN: A Unified Framework for Multi-label Image Classification. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; IEEE Computer Society: Los Alamitos, CA, USA, 2016; pp. 2285–2294.
68. Liu, Z.; Mao, H.; Wu, C.Y.; Feichtenhofer, C.; Darrell, T.; Xie, S. A ConvNet for the 2020s. *arXiv* **2022**, arXiv:2201.03545.
69. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
70. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning (PMLR), Long Beach, CA, USA, 10–15 June 2019; pp. 6105–6114.
71. Tolstikhin, I.O.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. MLP-mixer: An all-mlp architecture for vision. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 24261–24272.
72. Wightman, R. PyTorch Image Models. 2019. Available online: https://github.com/rwightman/pytorch-image-models (accessed on 8 March 2023).
73. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
74. Liu, Z.; Hu, H.; Lin, Y.; Yao, Z.; Xie, Z.; Wei, Y.; Ning, J.; Cao, Y.; Zhang, Z.; Dong, L.; et al. Swin Transformer V2: Scaling Up Capacity and Resolution. In Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 11999–12009. [CrossRef]
75. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
76. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
77. Chen, L.C.; Papandreou, G.; Schroff, F.; Adam, H. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv* **2017**, arXiv:1706.05587.
78. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Computer Vision—ECCV 2018*; Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 833–851.
79. Iakubovskii, P. Segmentation Models Pytorch. 2019. Available online: https://github.com/qubvel/segmentation_models.pytorch (accessed on 8 March 2023).
80. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 3431–3440. [CrossRef]
81. Sun, K.; Zhao, Y.; Jiang, B.; Cheng, T.; Xiao, B.; Liu, D.; Mu, Y.; Wang, X.; Liu, W.; Wang, J. High-Resolution Representations for Labeling Pixels and Regions. *arXiv* **2019**, *arXiv:1904.04514*.
82. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015*; Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 234–241.
83. Lin, T.; Goyal, P.; Girshick, R.; He, K.; Dollar, P. Focal Loss for Dense Object Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 318–327. [CrossRef]
84. Li, Y.; Xie, S.; Chen, X.; Dollar, P.; He, K.; Girshick, R. Benchmarking Detection Transfer Learning with Vision Transformers. *arXiv* **2021**, arXiv:2111.11429. https://doi.org/10.48550/arXiv.2111.11429.
85. Ismail Fawaz, H.; Lucas, B.; Forestier, G.; Pelletier, C.; Schmidt, D.F.; Weber, J.; Webb, G.I.; Idoumghar, L.; Muller, P.A.; Petitjean, F. InceptionTime: Finding AlexNet for time series classification. *Data Min. Knowl. Discov.* **2020**, *34*, 1936–1962. [CrossRef]
86. Rußwurm, M.; Pelletier, C.; Zollner, M.; Lefèvre, S.; Körner, M. BreizhCrops: A Time Series Dataset for Crop Type Mapping. *arXiv* **2020**, arXiv:1905.11893.
87. Rußwurm, M.; Körner, M. Multi-Temporal Land Cover Classification with Sequential Recurrent Encoders. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 129. [CrossRef]
88. Wang, Z.; Yan, W.; Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 1578–1585.
89. Tang, W.; Long, G.; Liu, L.; Zhou, T.; Jiang, J.; Blumenstein, M. Rethinking 1D-CNN for Time Series Classification: A Stronger Baseline. *arXiv* **2021**, arXiv:2002.10061.
90. Turkoglu, M.O.; D'Aronco, S.; Wegner, J.; Schindler, K. Gating Revisited: Deep Multi-layer RNNs That Can Be Trained. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 4081–4092. [CrossRef]
91. Pelletier, C.; Webb, G.I.; Petitjean, F. Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series. *Remote Sens.* **2019**, *11*, 523. [CrossRef]
92. Rußwurm, M.; Körner, M. Self-attention for raw optical Satellite Time Series Classification. *ISPRS J. Photogramm. Remote Sens.* **2020**, *169*, 421–435. [CrossRef]

93. Chen, H.; Chandrasekar, V.; Tan, H.; Cifelli, R. Rainfall Estimation From Ground Radar and TRMM Precipitation Radar Using Hybrid Deep Neural Networks. *Geophys. Rese. Lett.* **2019**, *46*, 10669–10678. [CrossRef]
94. Weng, Q.; Mao, Z.; Lin, J.; Guo, W. Land-Use Classification via Extreme Learning Classifier Based on Deep Convolutional Features. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 704–708. [CrossRef]
95. Castelluccio, M.; Poggi, G.; Sansone, C.; Verdoliva, L. Land Use Classification in Remote Sensing Images by Convolutional Neural Networks. *arXiv* **2015**, arXiv:1508.00092. https://doi.org/10.48550/arXiv.1508.00092.
96. Papoutsis, I.; Bountos, N.I.; Zavras, A.; Michail, D.; Tryfonopoulos, C. Efficient deep learning models for land cover image classification. *arXiv* **2022**, arXiv:2111.09451.
97. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
98. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; pp. 9992–10002. [CrossRef]
99. Scheibenreif, L.; Hanna, J.; Mommert, M.; Borth, D. Self-supervised Vision Transformers for Land-cover Segmentation and Classification. In Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), New Orleans, LA, USA, 19–20 June 2022; pp. 1421–1430. [CrossRef]
100. Zhang, C.; Wang, L.; Cheng, S.; Li, Y. SwinSUNet: Pure Transformer Network for Remote Sensing Image Change Detection. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 5224713. [CrossRef]
101. Wang, D.; Zhang, J.; Du, B.; Xia, G.S.; Tao, D. An Empirical Study of Remote Sensing Pretraining. *IEEE Trans. Geosci. Remote Sens.* 2022, *Early Access*. [CrossRef]
102. Liu, S.; He, C.; Bai, H.; Zhang, Y.; Cheng, J. Light-weight attention semantic segmentation network for high-resolution remote sensing images. In Proceedings of the IGARSS 2020—2020 IEEE International Geoscience and Remote Sensing Symposium, Waikoloa, HI, USA, 26 September–2 October 2020; pp. 2595–2598.
103. Xu, Z.; Zhang, W.; Zhang, T.; Yang, Z.; Li, J. Efficient Transformer for Remote Sensing Image Segmentation. *Remote Sens.* **2021**, *13*, 3585. [CrossRef]
104. Alhichri, H.; Alswayed, A.S.; Bazi, Y.; Ammour, N.; Alajlan, N.A. Classification of Remote Sensing Images Using EfficientNet-B3 CNN Model With Attention. *IEEE Access* **2021**, *9*, 14078–14094. [CrossRef]
105. Meng, Z.; Zhao, F.; Liang, M. SS-MLP: A Novel Spectral-Spatial MLP Architecture for Hyperspectral Image Classification. *Remote Sens.* **2021**, *13*, 4060. [CrossRef]
106. Gong, N.; Zhang, C.; Zhou, H.; Zhang, K.; Wu, Z.; Zhang, X. Classification of hyperspectral images via improved cycle-MLP. *IET Comput. Vis.* **2022**, *16*, 468–478. [CrossRef]
107. Solórzano, J.V.; Mas, J.F.; Gao, Y.; Gallardo-Cruz, J.A. Land Use Land Cover Classification with U-Net: Advantages of Combining Sentinel-1 and Sentinel-2 Imagery. *Remote Sens.* **2021**, *13*, 3600. [CrossRef]
108. Cheng, Z.; Fu, D. Remote Sensing Image Segmentation Method based on HRNET. In Proceedings of the IGARSS 2020—2020 IEEE International Geoscience and Remote Sensing Symposium, Waikoloa, HI, USA, 26 September–2 October 2020; pp. 6750–6753. [CrossRef]
109. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *40*, 834–848. [CrossRef]
110. Ramirez, W.; Achanccaray, P.; Mendoza, L.F.; Pacheco, M.A.C. Deep Convolutional Neural Networks for Weed Detection in Agricultural Crops Using Optical Aerial Images. In Proceedings of the 2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS), Santiago, Chile, 22–26 March 2020; pp. 133–137. [CrossRef]
111. Liu, M.; Fu, B.; Xie, S.; He, H.; Lan, F.; Li, Y.; Lou, P.; Fan, D. Comparison of multi-source satellite images for classifying marsh vegetation using DeepLabV3 Plus deep learning algorithm. *Ecol. Indic.* **2021**, *125*, 107562. [CrossRef]
112. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1*; MIT Press: Cambridge, MA, USA, 2015; pp. 91–99.
113. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
114. Zhong, L.; Hu, L.; Zhou, H. Deep learning based multi-temporal crop classification. *Remote Sens. Environ.* **2019**, *221*, 430–443. [CrossRef]
115. Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
116. Sun, Z.; Di, L.; Fang, H. Using long short-term memory recurrent neural network in land cover classification on Landsat and Cropland data layer time series. *Int. J. Remote Sens.* **2018**, *40*, 593–614. [CrossRef]
117. Ndikumana, E.; Ho Tong Minh, D.; Baghdadi, N.; Courault, D.; Hossard, L. Deep Recurrent Neural Network for Agricultural Classification using multitemporal SAR Sentinel-1 for Camargue, France. *Remote Sens.* **2018**, *10*, 1217. [CrossRef]
118. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 618–626.
119. Boguszewski, A.; Batorski, D.; Ziemba-Jankowska, N.; Zambrzycka, A.; Dziedzic, T. LandCover.ai: Dataset for Automatic Mapping of Buildings, Woodlands and Water from Aerial Imagery. *arXiv* **2020**, arXiv:2005.02264.

120. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollar, P.; Zitnick, L. Microsoft COCO: Common Objects in Context. In Proceedings of the Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, 6–12 September 2014; Springer: Cham, Switzerland, 2014.
121. Everingham, M.; Gool, L.V.; Williams, C.K.I.; Winn, J.; Zisserman, A. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.* **2009**, *88*, 303–308. [CrossRef]
122. Lu, X.; Zhang, Y.; Yuan, Y.; Feng, Y. Gated and Axis-Concentrated Localization Network for Remote Sensing Object Detection. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 179–192. [CrossRef]
123. Tan, M.; Le, Q.V. EfficientNetV2: Smaller Models and Faster Training. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Virtual Event, 18–24 July 2021; 2021; Volume 139, pp. 10096–10106.
124. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef]
125. Wang, Y.; Albrecht, C.; Ait Ali Braham, N.; Mou, L.; Zhu, X. Self-Supervised Learning in Remote Sensing: A Review. *IEEE Geosci. Remote Sens. Mag.* **2022**, *10*, 213–247. [CrossRef]
126. Akiva, P.; Purri, M.; Leotta, M. Self-Supervised Material and Texture Representation Learning for Remote Sensing Tasks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 8203–8215.
127. Manas, O.; Lacoste, A.; i Nieto, X.G.; Vazquez, D.; Rodriguez, P. Seasonal Contrast: Unsupervised Pre-Training from Uncurated Remote Sensing Data. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; IEEE Computer Society: Los Alamitos, CA, USA, 2021; pp. 9394–9403. [CrossRef]
128. Wang, Y.; Braham, N.A.A.; Xiong, Z.; Liu, C.; Albrecht, C.M.; Zhu, X.X. SSL4EO-S12: A Large-Scale Multi-Modal, Multi-Temporal Dataset for Self-Supervised Learning in Earth Observation. *arXiv* **2022**, arXiv:2211.07044.
129. Buslaev, A.; Iglovikov, V.I.; Khvedchenya, E.; Parinov, A.; Druzhinin, M.; Kalinin, A.A. Albumentations: Fast and flexible image augmentations. *Information* **2020**, *11*, 125. [CrossRef]
130. Oliphant, T.E. *A Guide to NumPy*; Trelgol Publishing: Austin, TX, USA, 2006; Volume 1.
131. Kramer, O. Scikit-learn. In *Machine Learning for Evolution Strategies*; Springer: Cham, Switzerland, 2016; pp. 45–53.
132. Szymanski, P.; Kajdanowicz, T. Scikit-multilearn: A scikit-based Python environment for performing multi-label classification. *J. Mach. Learn. Res.* **2019**, *20*, 209–230.
133. Bisong, E. Matplotlib and seaborn. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*; Springer: Berkeley, CA, USA, 2019; pp. 151–165.
134. zipp: A pathlib-Compatible Zipfile Object Wrapper. 2023. Available online: https://doc.sagemath.org/html/en/reference/spkg/zipp.html (accessed on 8 March 2023).
135. dill: Serialize All of Python. 2023. Available online: https://pypi.org/project/dill/ (accessed on 8 March 2023).
136. Lmdb: A Universal Python Binding for the LMDB 'Lightning' Database. 2023. Available online: https://lmdb.readthedocs.io/en/release/ (accessed on 8 March 2023).
137. tifffile: Storing NumPY Arrays in TIFF and Read Image and Metadata from TIFF-Like Files. 2023. Available online: https://pypi.org/project/tifffile/ (accessed on 8 March 2023).
138. h5py: A Pythonic Interface to the HDF5 Binary Data Format. 2023. Available online: https://www.h5py.org/ (accessed on 8 March 2023).
139. Click: Command Line Interface Creation Kit. 2023. Available online: https://click.palletsprojects.com/en/8.1.x/ (accessed on 8 March 2023).
140. Munch: A Dictionary Supporting Attribute-Style Access. 2023. Available online: https://morioh.com/p/bbdd8605be66 (accessed on 8 March 2023).
141. Marshmallow: Simplified Object Serialization. 2023. Available online: https://marshmallow.readthedocs.io/en/stable/ (accessed on 8 March 2023).
142. Detlefsen, N.S.; Borovec, J.; Schock, J.; Harsh, A.; Koker, T.; Liello, L.D.; Stancl, D.; Quan, C.; Grechkin, M.; Falcon, W. TorchMetrics—Measuring Reproducibility in PyTorch. *J. Open Source Softw.* **2022**, *7*, 4101. [CrossRef]
143. Sechidis, K.; Tsoumakas, G.; Vlahavas, I. On the Stratification of Multi-Label Data. In Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases—Volume Part III, Athens, Greece, 5–9 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 145–158.
144. Zhai, X.; Puigcerver, J.; Kolesnikov, A.; Ruyssen, P.; Riquelme, C.; Lucic, M.; Djolonga, J.; Pinto, A.S.; Neumann, M.; Dosovitskiy, A.; et al. A Large-scale Study of Representation Learning with the Visual Task Adaptation Benchmark. *arXiv* **2019**, arXiv:1910.04867.
145. Risojevic, V.; Stojnic, V. Do we still need ImageNet pre-training in remote sensing scene classification? *arXiv* **2021**, arXiv:2111.03690. [CrossRef]
146. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.