SINERGISE sentinelhub TOMTOM meteoblue Technical University of Munich SatCen

Global Earth Monitor

Deliverable 4.3

eo-learn gateways

The project has received funding from European Union's Horizon 2020 Research and Innovation Programme" under the Grant Agreement 101004112

PREPARATION SLIP			
	Name	**Organisation**	**Date**
Prepared by	Matej Batič	Sinergise	25.05.2021
Reviewed by	Devis Peressutti	Sinergise	28.05.2021
Approved for submission by	Mariza Pertovt	Sinergise	31.05.2021

EXECUTIVE SUMMARY

Executive summary text

In this deliverable we show the integration of `eo-learn` with popular machine learning and deep learning frameworks. In particular, we demonstrate the integration with conventional ML frameworks already in place, and is being used extensively.

From the numerous deep learning frameworks, we explain why we decided to interface `eo-learn` with two: TensorFlow and PyTorch. In order to avoid dependency, maintenance and implementation issues, we have decided that the gateways – interfaces between `eo-learn` and deep learning frameworks – will be developed as standalone packages. The integration with `eo-learn` and implementation of a number of deep learning models using TensorFlow framework has already been released in the `eo-flow` package. In the continuation of the Global Earth Monitor project, PyTorch interface will be added in a similar fashion, extending `eo-learn` workflows to another large deep learning community.

Contractual Delivery Date	31.05.2021
Actual Delivery Date	31.05.2021
Type of delivery	Demonstrator
Dissemination level:	Public

Global Earth Monitor

Table of Contents

1 Introduction 1

2 `eo-learn` 2

2.1 `eo-learn` overview 2

2.2 `eo-learn-io` data gateways 4

2.3 `eo-learn-ml-tools` sub-package 4

3 Machine Learning frameworks 6

3.1 `eo-flow` 8

3.1.1 Getting started with `eo-flow` 8

3.1.2 Writing custom code 9

3.1.3 Package structure 9

4 `eo-learn` and machine learning examples 10

4.1 Conventional ML 10

4.1.1 Land Use Land Cover classification example 10

4.1.2 Water bodies delineation example 11

4.2 Deep Learning 13

4.2.1 Field delineation example 13

4.2.2 Tree cover prediction example 14

4.2.3 Crop type classification example 16

4.2.4 Super Resolution using Sentinel-2 and Deimos imagery example 19

5 Conclusions 22

List of Figures

Figure 1: eo-learn as a bridge 2
Figure 2: EOPatch data structure overview..... 3
Figure 3: Data collections available through Sentinel Hub service..... 4
Figure 4: LULC predictions over 1224 EOPatches covering Azerbaijan. 4
Figure 5: A temporal stack of Sentinel-2 images is needed to predict land cover. 10
Figure 6: Sentinel-2 image (left), ground truth (centre) and prediction (right)..... 11
Figure 7: Confusion matrix visualises the performance of an algorithm..... 11
Figure 8: EOTasks for waterbody extent delineation workflow. 12
Figure 9: Laguna Ralicó in Argentina. 12
Figure 10: Example of agricultural parcel boundary predictions overlaid to a Sentinel-2 image. 13
Figure 11: Web application for interactive exploration of field delineation results. 14
Figure 12: RGB image of an EOPatch on the left, and ground truth - EU tree cover density for 2015 15
Figure 13: RGB patchlet (left), tree cover from ground truth (middle) and prediction (right). 15
Figure 14: Confusion matrix for tree cover prediction. 16
Figure 15: Crop type classification example shows the pipeline over an area in Austria. 17
Figure 16: RGB plot of one Sentinel-2 observation (left), ground truth data over same area (right)..... 17
Figure 17: Confusion matrices for random forest (left) and TCNN (right) approach..... 18
Figure 18: Comparison of RF prediction (left) and ground truth (right) data. 19
Figure 19: HighResNet architecture, image from reference paper. 20
Figure 20: Comparison of Sentinel-2 and super-resolved image. 20

List of Tables

Table 1: Python libraries and frameworks for ML that are used within eo-learn..... 7

Table 2: Scores per crop type for random forest (ML) and TCNN (DL) approach. 18

List of Abbreviations

aDC	Adjustable Data Cube
AOI	Area Of Interest
BOA	Bottom of the Atmosphere
CV	Computer Vision
DL	Deep Learning
DS	Data Science
EO	Earth Observation
ESA	European Space Agency
GEM	Global Earth Monitor
GSD	Ground Sampling Distance
LULC	Land Use / Land Cover
MFSR	Multi-Frame Super-Resolution
ML	Machine learning
NDWI	Normalized Difference Water Index
RGB	Red, Green, Blue
RS	Remote Sensing
SR	Super-Resolution
TCNN	Temporal Convolutional Neural Network
VHR	Very High Resolution

1 Introduction

The availability of open and commercial Earth Observation (EO) data is growing at unprecedented rate. Consequently, the many EO applications, ranging from land use and land cover (LULC) monitoring, crop monitoring and yield prediction, to disaster control, emergency services and humanitarian relief have more and more data to process in order to automatically extract complex patterns in such spatio-temporal data. Tools that make analyses of EO data easy, scalable and fast are in high demand.

eo-learn is an open-source Python library that acts as a bridge between Earth Observation/Remote Sensing and the Python ecosystem for data science (DS) and machine learning (ML). `eo-learn` aims at making an entry to the field of Remote Sensing (RS) for non-experts easier, while at the same time bringing the state-of-the-art tools for computer vision, machine learning, and deep learning (DL) existing in the Python ecosystem to RS experts. `eo-learn` is a by-product of the Perceptive Sentinel Horizon 2020 project and has received a lot of traction and usage within the EO/RS community. `eo-learn` continues to grow within GEM, with particular focus on bridging the data to ML frameworks.

In Section 2, an introduction to `eo-learn` will be given, particularly to the various gateways to obtaining data. It will be followed by Section 3 with a short outline of existing ML frameworks, reporting on why only a subset of frameworks was selected. Section 4 will demonstrate the usage of eo-learn within ML frameworks for both shallow and deep learning. In the conclusion Section 5, we will outline future plans, with emphasis on supporting GEM use-cases.

2 eo-learn

eo-learn is a framework for working with EO data. Technically, it is a collection of open-source Python packages allowing a user to develop a data-analysis workflow that includes different EO/RS tasks, such as cloud masking, image co-registration, feature extraction, classification, etc.... For example, a user could use eo-learn to predict land-use based on machine-learning techniques or determine water levels of a water body (dam, reservoir, lake, etc.) from satellite imagery. With its modular design, eo-learn is easy to use and encourages collaboration by sharing and reusing tasks used in typical EO-value-extraction workflows. eo-learn is openly shared under MIT license, with very limited restriction on reuse, therefore making eo-learn noteworthy to a wider audience (e.g., academic community, commercial entities). The code is available on GitHub:

https://github.com/sentinel-hub/eo-learn

Since its inception, eo-learn has had 23 releases and has been installed more than 40 thousand times (the statistics up to May 2021). The repository on GitHub has 740 stars, 223 forks (copies of a repository on GitHub) and 27 contributors from 13 entities (companies, academic institutes, etc.) with almost 1700 commits. A large amount of effort has gone into documenting the features of the packages, providing examples and helping users with issues. The documentation is available on:

https://eo-learn.readthedocs.io/en/latest/

While some users rely on GitHub for providing feedback (through GitHub issues), the user support is also available on Sentinel-Hub forum:

https://forum.sentinel-hub.com/c/apps-services/python-sdk/23

2.1 eo-learn overview

eo-learn is subtitled a "bridge between EO and Python ecosystem for data science and ML", which is often represented with the Figure 1.

Figure 1: eo-learn as a bridge between EO and other geo-spatial data (on left) and Python data science ecosystem (on right).

eo-learn is divided into several sub-packages according to different functionalities and external package dependencies.

- `eo-learn-core` - The main sub-package which implements basic building blocks (`EOPatch`, `EOTask` and `EOWorkflow`) and commonly used functionalities.
- `eo-learn-io` - Input/output sub-package that deals with obtaining data.
- `eo-learn-mask` - The sub-package used for masking of data and calculation of cloud and snow masks.
- `eo-learn-geometry` - Geometry sub-package used for geometric transformation and conversion between vector and raster data.
- `eo-learn-features` - A collection of utilities for extracting data properties and feature manipulation.
- `eo-learn-ml-tools` - Various tools that can be used before or after the machine learning process.
- `eo-learn-coregistration` - The sub-package that deals with image co-registration.
- `eo-learn-visualization` - Visualisation tools for core elements of `eo-learn`.

The building blocks of `eo-learn` are `EOPatch`, `EOTask` and `EOWorkflow` objects from the `eo-learn-core` sub-package. All data are stored in `EOPatch` instances, where dictionaries store raster data as NumPy arrays and vector data as Shapely geometries for both temporal as well as time-independent spatial and scalar information, as shown in Figure 2. An `EOPatch` instance is uniquely defined by coordinates of a bounding box and possibly the time-interval the stored data refers to. Information in any format readable by Python packages can also be stored in `EOPatch` objects.

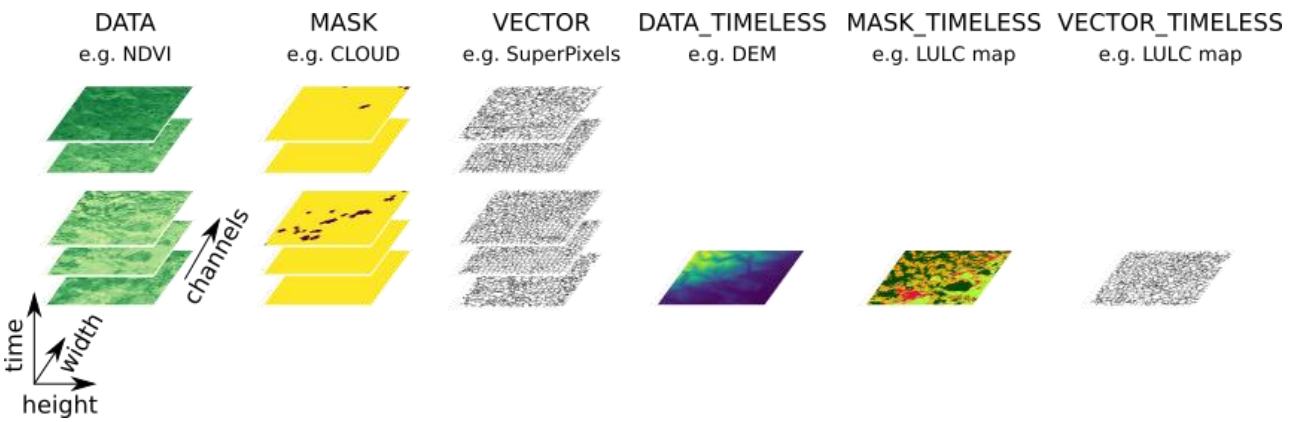

Figure 2: EOPatch data structure overview.

Any operation on `EOPatch` instances is performed by `EOTask` instances, which can be inter-connected into a `EOWorkflow` to build a complete pipeline. `EOWorkflow` allows definition of a workflow in the form of an acyclic graph, where `EOTask` instances are vertices of the graph and `EOPatch` instances flow through the edges connecting the vertices. `EOWorkflow` can be run in parallel to different input `EOPatch` instances, allowing automatic processing of large amounts of *spatio-temporal* data.

In the following sections we will shortly describe two sub-packages: `eo-learn-io` as a gateway to data, and `eo-learn-ml-tools` as a (simple) gateway to ML frameworks.

2.2 `eo-learn-io` data gateways

To get the EO data `eo-learn` heavily relies on Sentinel Hub services, accessing it through the sentinelhub-py Python library. This gives users straight-forward access to all data collections available through Sentinel Hub, both publicly available EO data (e.g., Sentinels, Landsat), products (e.g., Global Land Cover by Copernicus Land Service, Global Surface Water dataset by the European Commission's Joint Research Centre and others), as well as users' own commercial datasets and other raster data imported to Sentinel Hub with "Bring your Own Data" capabilities. A broad illustration of data, available through Sentinel Hub services, is shown in Figure 3.

Figure 3: Data collections available through Sentinel Hub service.

At the same time, `eo-learn-io` sub-package contains task that allow users interaction with their own (locally) stored data as well; the `ImportFromTiff` task imports data from a GeoTiff file into an `EOPatch` structure. From that point onwards, tasks from `eo-learn` can be used in the same way as if the data was loaded from the Sentinel Hub service.

2.3 `eo-learn-ml-tools` sub-package

Figure 1 shows `eo-learn` as a bridge, where the side of the bridge facing EO data is covered by the `eo-learn-io` gateway. `eo-learn` was designed with an idea in mind to work with the most popular open ML frameworks, however real integration has not been realized up to date. The `eo-learn-ml-tools` sub-package contains various tasks and utility methods that can be used before or after the machine learning process, representing the other gateway: to ML ecosystems. At the moment, the sub-package contains algorithms limited to pixel-based approaches, and does not yet support causal analysis of time-series. Nevertheless, it has shown great value, and was used to produce LULC maps over large areas, e.g., over Azerbaijan, as shown in Figure 4.

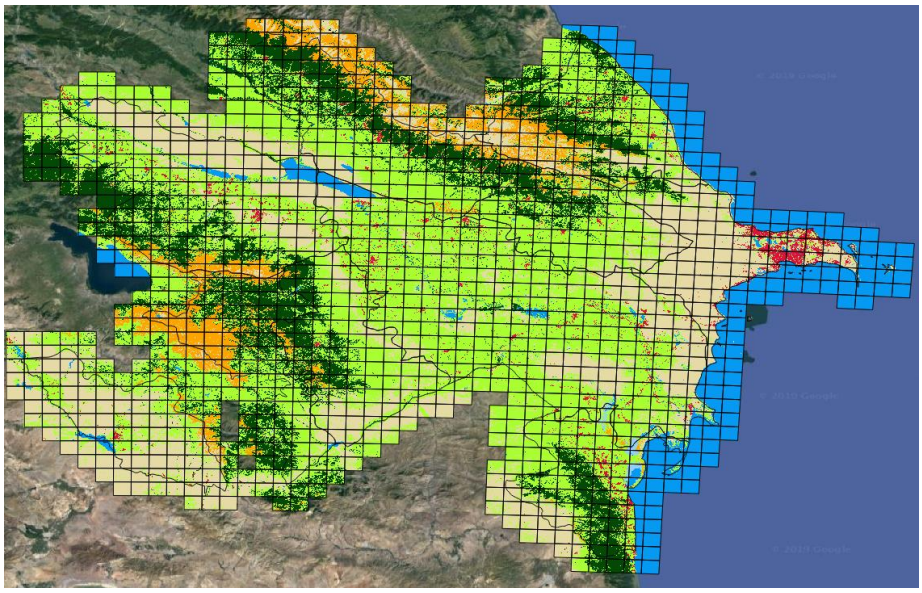

Figure 4: LULC predictions over 1224 EOPatches covering Azerbaijan.

How `eo-learn` can be used to produce a LULC map as shown in Figure 4 is the main topic of a series of blog posts (part 1, part 2, and part 3) and notebooks in the eo-learn examples. These blog-posts have been read almost 30 thousand times, showing the reach and interest of both EO and ML communities in the topic.

The following section will dig a bit deeper into the ML Python ecosystems, and how `eo-learn` approaches the gateways to ML frameworks.

3 Machine Learning frameworks

`eo-learn` is a data science (DS) tool that caters to various EO projects and workflows. As in other fields of DS, the predictive capability of ML algorithms is an important aspect of EO and RS. From the point of the information it provides, satellite imagery is pre-defined as it provides detector responses within pre-defined intervals in electro-magnetic spectrum, and the (pre-)processing steps define how this information is seen (by a machine). Once this pre-processing is done (`eo-learn` tasks already cover many common pre-processing workflows), the data has to be fed into the appropriate ML algorithm. Unfortunately, the vast availability of ML libraries accessible within Python ecosystem makes it difficult to incorporate all options into `eo-learn` without encountering dependency, interoperability and maintenance issues. So far, we have limited `eo-learn` interaction to few most popular libraries and frameworks, seen in Table 1.

Table 1 clearly shows that currently, `eo-learn` is well coupled with conventional (shallow) ML algorithms, focused to pixel-based approaches. Section 4.1 demonstrates examples of such conventional approaches, which have been very well accepted in the EO community, and continue to be the state-of-the-art in many applications because of their ease of use, scalability, speed and performance.

Within work package *4: Machine learning*, the development and adoption of new ML techniques will be tested on project use-cases. The development of new approaches is primarily driven by recent advances from the research community of ML, DS, and computer vision (CV). ML methods which will prove useful for GEM use-cases will be considered for integration into `eo-learn`. The integration of `eo-learn` and deep learning frameworks is thus one of the main objectives of the GEM project.

Unfortunately, there are many DL frameworks available within Python ecosystem, with many similar but also different utilities. There is no clear favourite framework in an absolute sense, but depending upon the requirements, one framework may offer better functionalities than others. This poses a problem of which framework to choose, as supporting several comes with same risks as mentioned before: dependency, interoperability and maintenance issues. To limit the scope and increase the possibilities of success, we have investigated which DL frameworks are best suited, and particularly which are used by the partners in the GEM consortium.

Already in 2019, the KDNuggets poll showed that TensorFlow, Keras and PyTorch covered 70% of all DL usage. We additionally considered also Apache MXNet, and Theano, but did not continue investigating their suitability. Theano is not actively developed anymore, and Apache MXNet has a rather small community support and is substantially less popular in the research community.

Keras is widely known for being the most simplistic and easy to use neural network (NN) framework out there, while TensorFlow is better suited for building a production scale model, offering the best community support. For DL research, PyTorch provides the best functionalities and flexibility. Since Keras has become the recommended API for TensorFlow since version 2.0, we really just have two contestants: TensorFlow and PyTorch.

Table 1: Python libraries and frameworks for ML that are used within eo-learn.

Library / framework	description
numpy	The fundamental package for scientific computing with Python. Offers comprehensive set of mathematical functions operating on multi-dimensional data arrays. All raster data within `eo-learn` are handled as numpy arrays and majority of `EOTask` implementations manipulate said arrays to perform some specific task, e.g. temporal interpolation. Some examples of using numpy in `eo-learn`: • NormalizedDifferenceIndexTask The task calculates a Normalized Difference Index (NDI) between two bands A and B • InterpolationTask Main `EOTask` for interpolation and resampling of time-series.
pandas and geopandas	pandas is a library providing high-performance, easy-to-use data structures and data analysis tools. Within `eo-learn` geopandas (geo "extension" to pandas) is used for handling vector data. Some examples of using pandas and geopandas in `eo-learn`: • geometry io tasks These tasks can import vector data from several sources, and write data to `EOPatch` in `GeoDataFrame` format
SciPy	SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. It has different modules for optimization, linear algebra, integration and statistics and is particularly useful for image (raster data) manipulations. Some examples of SciPy in `eo-learn`: • DoublyLogisticApproximationTask An `EOTask` for calculation of doubly logistic approximation on each pixel of a feature in `EOPatch`. • HaralickTask Task to compute Haralick texture images; the grey-level co-occurrence matrix (GLCM) on a sliding window over the input image and extract the texture properties into new feature. • LocalBinaryPatternTask `EOTask` looks at points surrounding a central point and tests whether the surrounding points are greater or less than the central point.
Scikit-Learn	Scikit-Learn is an open-source ML framework, providing tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities. Scikit-learn library can be used with the following ML tasks: classification, clustering, regression, factor analysis, principal component analysis, model selection, pre-processing, and dimensionality reduction. Scikit-Learn also provides model analysis tools like the confusion matrix for assessing how well a model performed. Scikit-learn is focussed on data modelling, making it a perfect companion to `eo-learn`, which handles data loading, manipulation and visualization. Some examples of using Scikit-Learn in `eo-learn`: • ClusteringTask Tasks computes clusters on selected features
LightGBM	LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient, offering faster training speed, higher efficiency and lower memory usage to reach higher accuracy. It is particularly well suited to handle large-scale data. LightGBM is used in the Land Use / Land Cover Prediction for Slovenia example within `eo-learn`.

TensorFlow was first developed by Google Brain and later open sourced. It is by far the largest framework in terms of the user base and community support. With TensorFlow one can create and train ML models also on mobile devices and high-performance servers by using TensorFlow Lite and TensorFlow Serving. PyTorch was developed by Facebook AI Research. It is deemed easy to learn, use and integrate with the rest of the Python ecosystem, is very customizable, and widely used in DL research.

Even after consulting with consortium partners, there is no clear winner, so for the time being we are operating under the assumption that `eo-learn` will be supporting both of these two frameworks. Due to risk of exploding dependencies and possible dependency conflicts, the interface between `eo-learn` and particular framework will be in a separate and dedicated package. Each package for the given framework will implement data loaders for EO datasets and the most common DL models for EO applications (e.g., time-series classification, semantic segmentation).

In the next section we will describe and demonstrate the first of such interfaces: `eo-flow`.

3.1 `eo-flow`

As you might have guessed from the name, `eo-flow` is an interface between `eo-learn` and TensorFlow. It is available at

https://github.com/sentinel-hub/eo-flow

The repository provides code and examples for creation of EO projects using TensorFlow. The code uses TensorFlow 2.0 with Keras as the main model building API. Common model architectures, layers, and data loaders for EO data are provided in the `eoflow` package. Custom models and input methods can also be implemented building on top of the provided abstract classes. The package aims at seamless integration with eo-learn, and favours both creation of models for prototyping as well as production of EO applications.

The package can be installed by running the following command:

```
$ pip install git+https://github.com/sentinel-hub/eo-flow
```

Users can also install the package from source. To do so, one has to clone the repository and run the following command in the root directory of the project:

```
$ pip install .
```

3.1.1 Getting started with `eo-flow`

The `eoflow` package can be used in two ways. For best control over the workflow and faster prototyping, the package can be used programmatically (in code). The example notebook should help users to get started with that. It demonstrates how to prepare a dataset pipeline, train the model, evaluate the model and make predictions using the trained model.

An alternate way of using `eoflow` is by writing configuration `json` files and running them using `eoflow`'s execute script. Configuration files specify and configure the task (training, evaluation, etc.) and contain the configurations of the model and input methods. Example configurations are provided in the `configs` directory. Once a configuration file is created it can be run using the execute command from the command line interface (CLI), allowing automation of training and testing of models.

A simple example can be run in CLI using the following command. More advanced configurations are also provided.

```
$ python -m eoflow.execute configs/example.json
```

This will create an output folder `temp/experiment` containing the TensorBoard logs and model checkpoints.

To visualize the logs in TensorBoard, run

```
$ tensorboard --logdir=temp/experiment
```

3.1.2 Writing custom code

To get started with writing custom models and input methods for `eoflow`, users are encouraged to take a look at the example implementations (examples folder). Custom classes use schemas to define the configuration parameters in order to work with the execute script and configuration files. Since `eoflow` builds on top of TensorFlow 2.0 and Keras, model building is very similar.

3.1.3 Package structure

The sub-packages of `eoflow` are as follows:

- `base`: this directory contains the abstract classes to build models, inputs and tasks. Any useful abstract class should go in this folder.
- `models`: classes implementing the TensorFlow models (e.g., Fully-Convolutional-Network, GANs, seq2seq, etc.). These classes inherit and implement the `BaseModel` abstract class. The module also contains custom losses, metrics and layers.
- `tasks`: classes handling the configurable actions that can be applied to each TF model, when using the execute script. These actions may include training, inference, exporting the model, validation, etc. The tasks inherit the `BaseTask` abstract class.
- `input`: building blocks and helper methods for loading the input data (`EOPatch`, numpy arrays, etc.) into a TensorFlow Dataset and applying different transformations (data augmentation, patch extraction)
- `utils`: collection of utility functions

State-of-the-art architectures and examples for land cover, crop classification and semantic segmentation using satellite imagery and time-series are provided.

4 eo-learn and machine learning examples

The previous section has described the current interfaces of eo-learn and ML frameworks. This section will present examples of using ML and DL methods with eo-learn. Conventional ML approaches will be shown first, since they are more tightly integrated within the library. More recent and complex DL approaches, mostly using eo-flow, will be presented later on in Section 0.

4.1 Conventional ML

This section reports examples of conventional ML methods applied to EO applications. In this context, conventional (i.e., shallow) ML refers to algorithms that model (non-)linear dependencies between input features and output target properties, such as land classes and crop cultures. The input features are typically hand-crafted and designed by the developers including domain knowledge. Example of conventional ML algorithms can be found in the scikit-learn examples.

4.1.1 Land Use Land Cover classification example

The notebook example in eo-learn shows the steps towards constructing a machine learning pipeline for predicting the land use and land cover for the region of Republic of Slovenia. The example uses features extracted from satellite images acquired by ESA's Sentinel-2 satellite to train a model and use it for land cover prediction. The example leads the reader through the entire process of creating the pipeline, with details provided at each step. The topic is also covered in a series of blog posts. First two parts (part 1 and part 2) describe how eo-learn and Random Forest (RF) from LightGBM framework can be jointly used to predict 10 land cover classes.

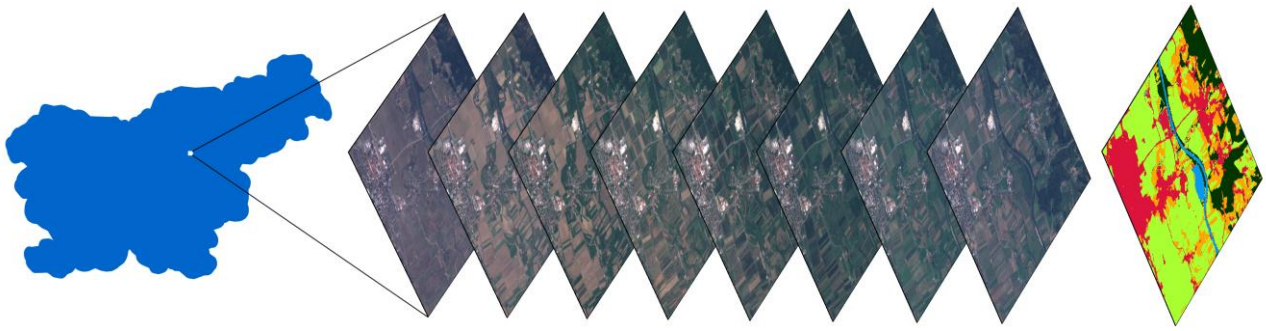

Figure 5: A temporal stack of Sentinel-2 images is needed to predict land cover.

Blog posts and corresponding notebook example in the eo-learn are amongst the most popular posts in our portfolio, and based on the number of related questions on the Sentinel Hub forum, also used over a number of different areas (countries).

RF classifier consists of an ensemble of decision trees, where the ensemble classification is performed using majority voting of the single tree predictions. Although decision trees are considered simplistic, the ensemble approach is robust, not prone to overfitting and provides satisfactory results being substantially less computationally expensive compared to DL models. As we can see from Figure 6, the classifier correctly recognises the landing strip as grassland, which is marked as artificial surface in the official land use data.

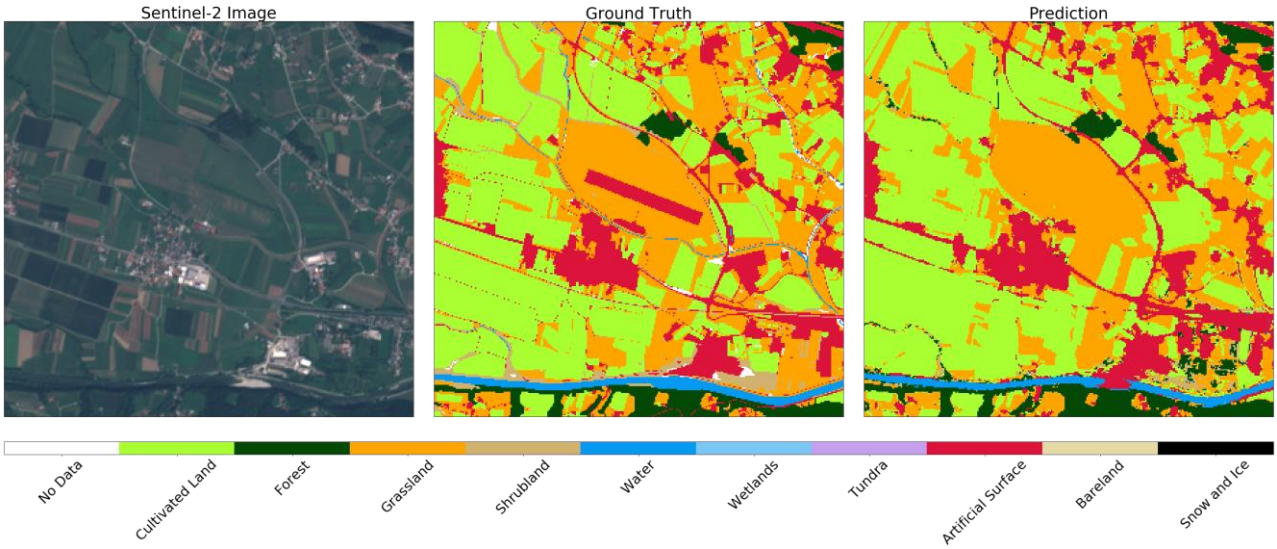

Figure 6: Sentinel-2 image (left), ground truth (centre) and prediction (right) for the area around the small sports airfield Levec, near Celje, Slovenia. The classifier correctly recognises the landing strip as grassland, which is marked as artificial surface in the official land use data.

The confusion matrix plot in Figure 7 shows that for most of the classes the model seems to perform rather well, which is another reason for this approach to be so popular.

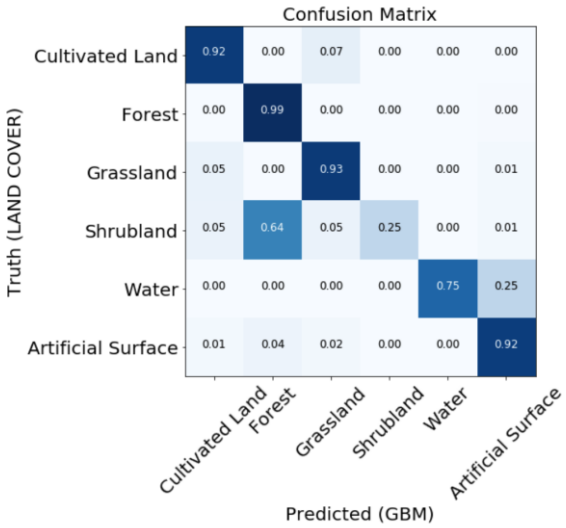

Figure 7: Confusion matrix visualises the performance of an algorithm: ground truth is shown in rows while each column represents the instances of the predicted class.

4.1.2 Water bodies delineation example

The water monitor workflow example in `eo-learn` showcases an EO processing chain that determines water levels of an open water body (dam, reservoir, lake, etc.) from Sentinel-2 imagery. The input is a polygon with water body's nominal water extent and the entire processing chain is performed using the `eo-learn` package, resulting in a vectorised extent of the water body.

The `EOWorkflow` for the water body delineation is shown in Figure 8. The thresholding of the normalised difference water index (NDWI) is done with `Scikit-Learn` implementation of Otsu's method.

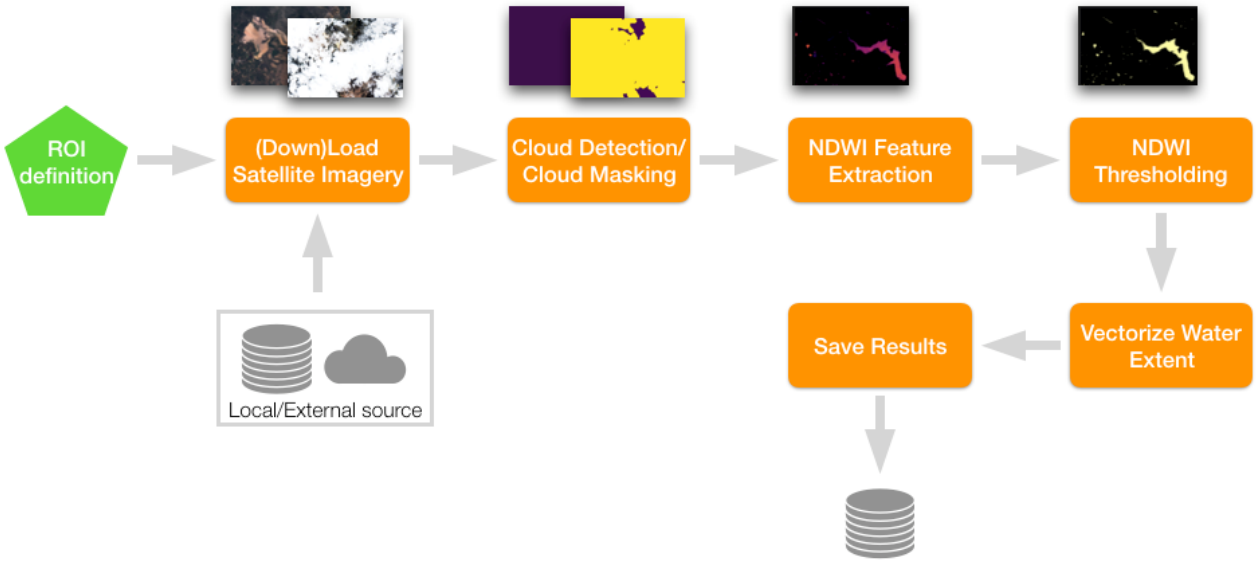

Figure 8: EOTasks for waterbody extent delineation workflow.

The approach presented in the example is used in the [Bluedot Observatory](), which implements automatic processing of about 13000 waterbodies globally for each cloudless Sentinel-2 observation. The publicly available [dashboard]() thus holds almost 2 million measurements of water body extents from the whole archive of Sentinel-2 imagery. Figure 9 shows the historical water extents for Laguna Ralicó in Argentina.

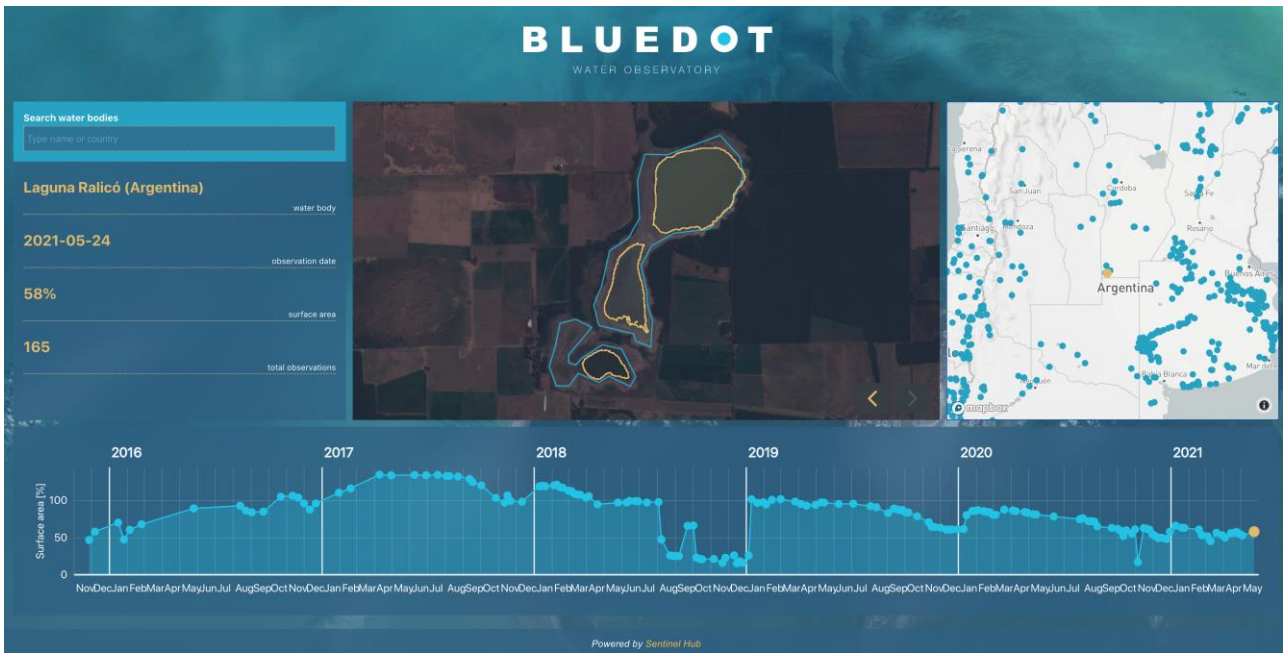

Figure 9: Laguna Ralicó in Argentina. Interactive dashboard allows users to move through time and view satellite imagery for each observation of water levels.

4.2 Deep Learning

This section reports examples of DL methods applied to EO applications. DL methods model the (non-)linear dependencies between input data and output target properties using more complex algorithms than conventional ML, typically based on Neural Networks. DL algorithms can use millions of parameters to model complex dependencies, and are typically trained iteratively through gradient optimization of an appropriate loss function.

4.2.1 Field delineation example

`eo-flow` has been used as integration of `eo-learn` and TensorFlow to develop an automatic system for parcel boundary delineation based on Sentinel-2 imagery. The delineated boundaries can aid the farmers speed up the declaration process and the paying agencies to better monitor changes in agricultural use. The development of the system was done by Sinergise within "New IACS Vision in Action" -- NIVA H2020 project. The repository contains code to generate automatic contours for agricultural parcels, given Sentinel-2 images, and has been used to generate contours for Lithuania and provinces in Spain.

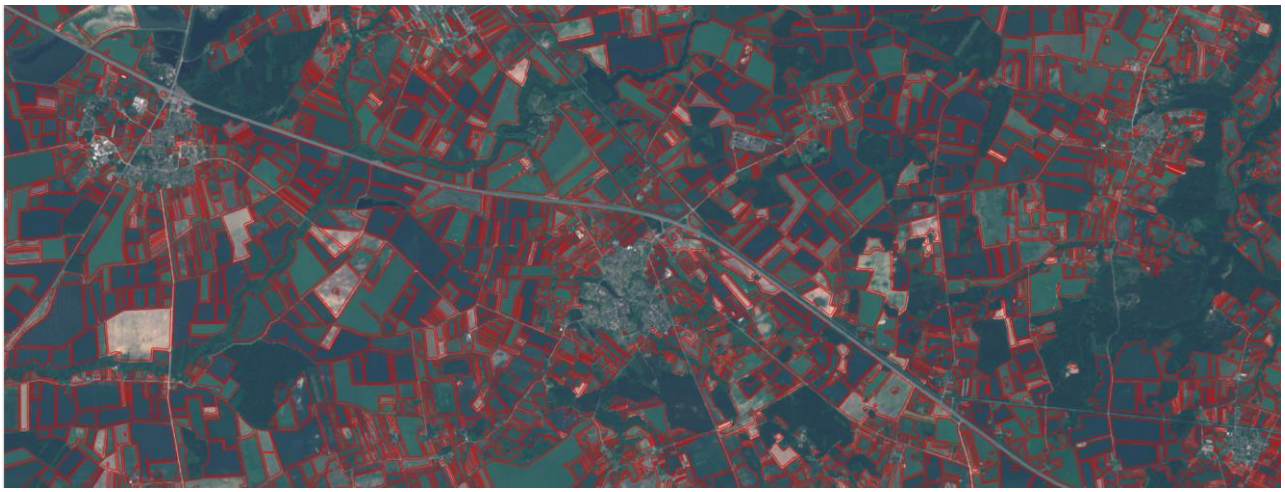

Figure 10: Example of agricultural parcel boundary predictions overlaid to a Sentinel-2 image.

Different models were tested, building on top of a *vanilla* U-net architecture, with each version adding components building up to the architecture proposed in Waldner *et al.*[1] Different components (e.g., residual convolutions, pyramid pooling) were implemented and tested separately to understand their influence on the results. The components and the final architecture can be found in `eo-flow` library, together with the implementations of metrics and loss functions. The blog post Parcel boundary detection for CAP describes the workflow composed of obtaining the satellite data, adding training data, preparing the data for DL model, followed by train and evaluation of the model, to final prediction and post-processing of the results.

For demonstration purposes users can visualise the final results of the workflow on top of Sentinel-2 imagery through web application parcelio.

[1] Waldner, F. & Diakogiannis, F. I. Deep learning on edge: extracting field boundaries from satellite images with a convolutional neural network. arXiv.org (2019).

Figure 11: Web application for interactive exploration of field delineation results.

4.2.2 Tree cover prediction example

The example notebook in `eo-learn` presents a toy example for training a deep learning architecture for semantic segmentation of satellite images using `eo-learn` and `keras`. The example showcases tree cover prediction over an area in France. The ground-truth data is retrieved from the EU tree cover density (2015) through Geopedia.

The workflow is as follows:

- input the area-of-interest (AOI)
- split the AOI into small manageable `EOpatches`
- for each `EOPatch`:
  - download red, green and blue (RGB) bands from Sentinel-2 Level-2A products providing bottom of the atmosphere (BOA) reflectance
  - retrieve corresponding ground-truth from Geopedia using a WMS request
  - compute the median values for the RGB bands over the time-interval
  - save to disk
  - select a 256x256 patch with corresponding ground-truth to be used for training/validating the model
- train and validate a U-net.

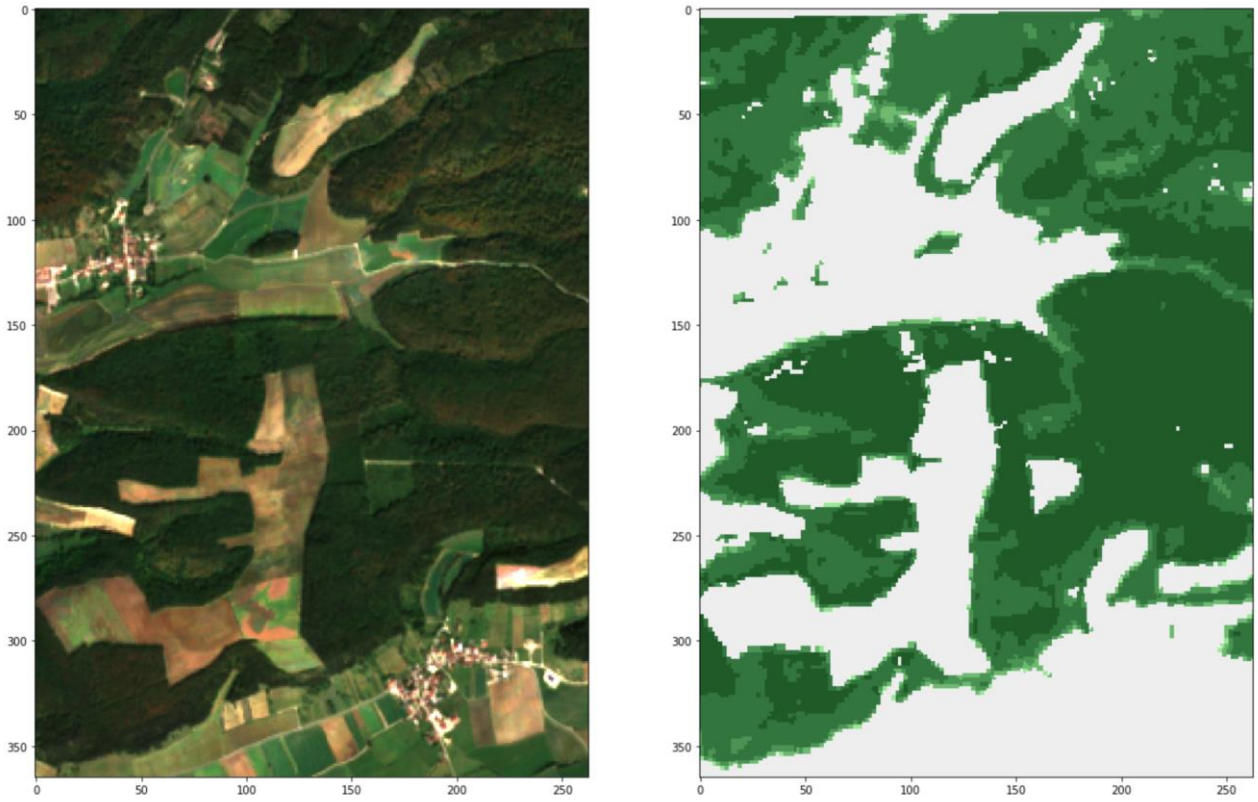

Figure 12: RGB image of an EOPatch on the left, and ground truth - EU tree cover density for 2015 - on the right. The RGB image is brightened to better show the forest areas.

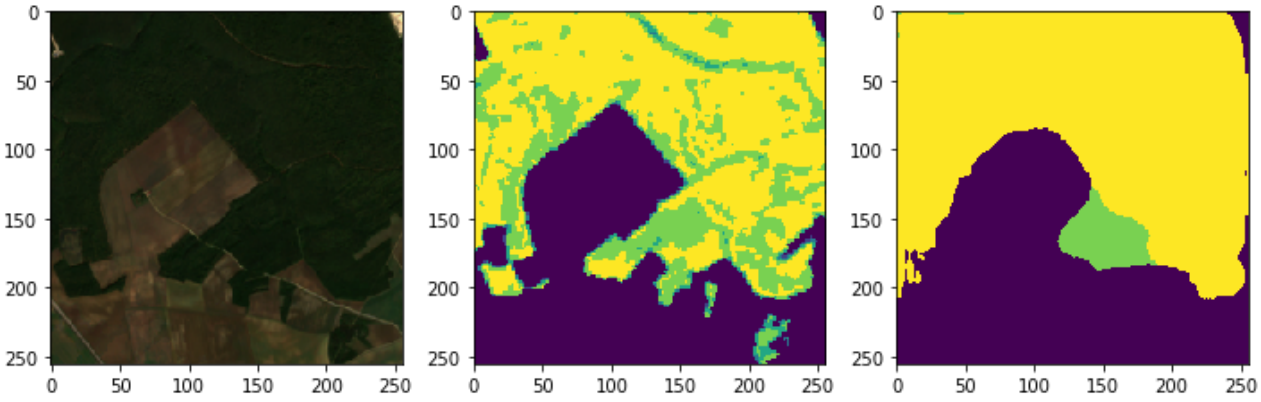

Figure 13: RGB patchlet (left), tree cover from ground truth (middle) and prediction (right).

Despite good quality input data (albeit old), the model learns mostly to distinguish areas completely free of trees and areas with 100% tree cover. The classes in-between are more difficult to discern, as can be seen from the confusion matrix in Figure 14. On the other hand, the use of limited number of data (only three bands) makes this approach somewhat lightweight.

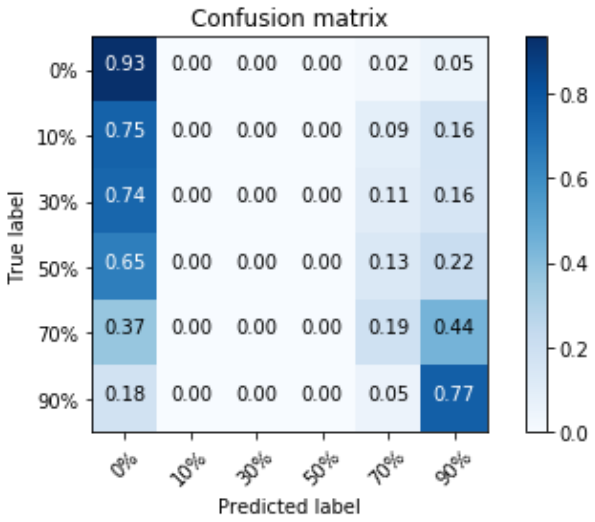

Figure 14: Confusion matrix for tree cover prediction.

The code in the example is planned to be upgraded to use U-net model implemented in `eo-flow` in the forthcoming months with the main benefit of reducing the complexity of the notebook.

4.2.3 Crop type classification example

The example notebook shows the steps towards constructing an automated ML pipeline for crop type identification for an area of interest. Along with the example, two different ML approaches are applied and compared. The first one, the LightGBM, represents a state-of-the-art conventional ML algorithm. The second one is a Temporal Convolutional Neural Network (T-CNN) architecture from the field of DL, where the prediction is performed on a time-series of Sentinel-2 scenes from 2018.

The example notebook leads the reader through the whole process of creating the pipeline, from four main processing blocks, i.e., `EOWorkflows`: (i) ground truth data, (ii) EO data, (iii) feature engineering + crop type grouping + sampling and lastly, (iv) prediction. The last part consists of several components: setting up and training each model (ML and DL), model validation and evaluation, prediction and finally visualization of results.

The AOI in this example covers a small region around Wels in Austria, as can be seen in Figure 15. The first two steps fill the `EOPatches` with both ground truth and EO data, as can be seen in Figure 16. Figure 16 also shows that crop types in the AOI are very diverse; each colour stands for one of the over 200 classes. The third step prepares the data for training, e.g., fixes some mistakes from ground truth data, groups the crops, spatially erodes the field polygons to reduce mixing from borders of agricultural fields, evenly samples pixels from `EOPatches`, and finally prepares training and testing dataset.

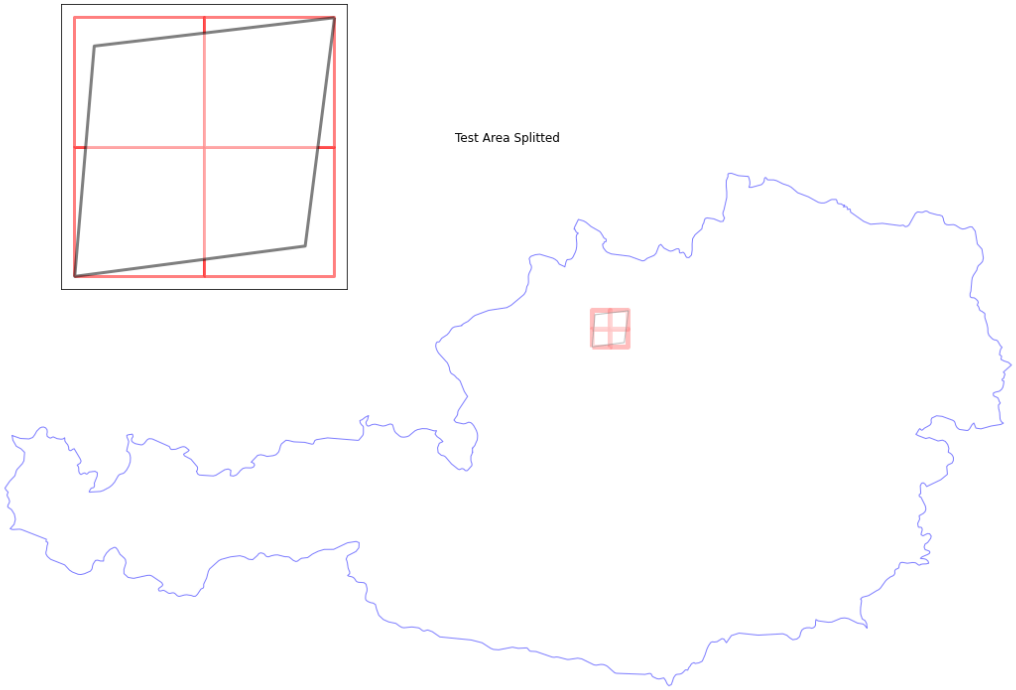

Figure 15: Crop type classification example shows the pipeline over an area in Austria.

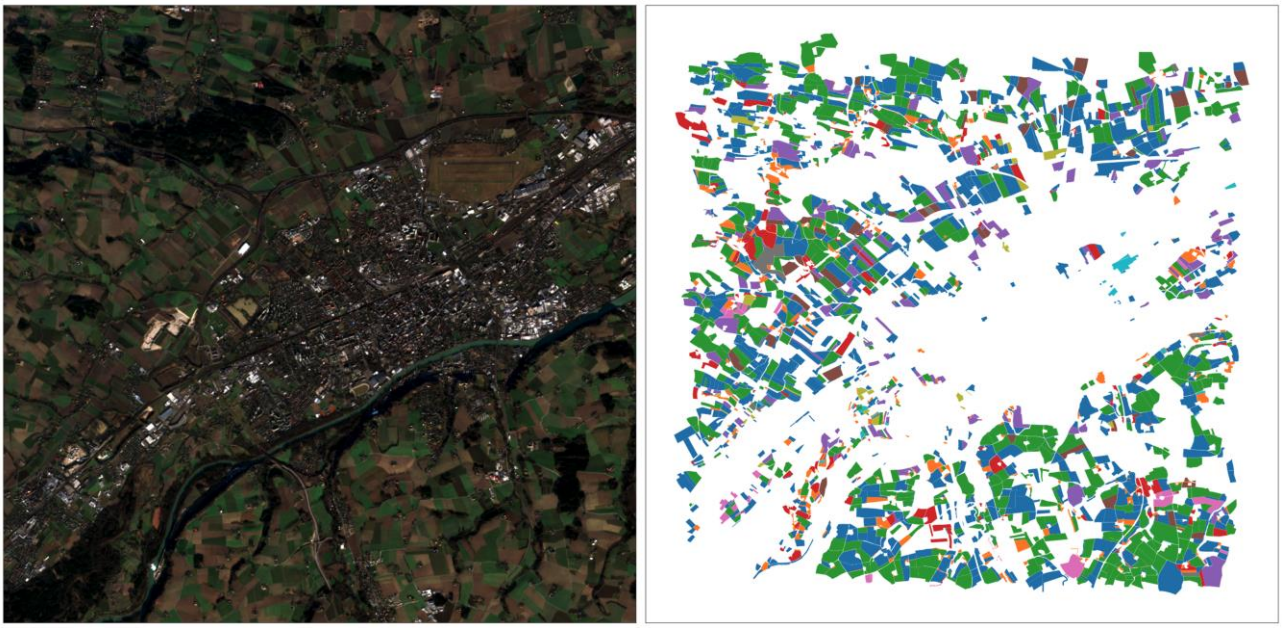

Figure 16: RGB plot of one Sentinel-2 observation (left), ground truth data over same area (right).

The two architectures (and two frameworks) are subsequently evaluated. The RF model is created using LightGBM while the TempCNN DL architecture uses convolutional Neural Networks (CNNs). So far CNNs were mainly and successfully applied for image processing tasks. However, convolutional filters can be applied to 1D signals as in the Temporal CNN, therefore successfully exploiting the temporal information of satellite image time series. The F_1 scores, recall and precision for both approaches are given in Table 2. The confusion matrices are shown in Figure 17.

Table 2: Scores per crop type for random forest (ML) and TCNN (DL) approach.

	Random Forest			Temporal CNN		
Crop type	F_1	Recall	Precision	F_1	Recall	Precision
Peas	89.1	96.6	82.7	87.5	96.8	79.8
Grass	75.4	86.0	67.2	67.1	65.9	68.3
Winter rape	97.4	95.0	100.0	97.4	100.0	94.8
Maize	92.9	94.6	91.2	88.1	95.5	81.8
Winter cereals	94.7	99.8	90.2	92.8	99.8	86.7
Pumpkins	67.7	59.4	78.6	76.1	80.4	72.2
Summer cereals	69.6	71.2	68.1	66.2	58.0	77.0
Vegetables	72.5	67.9	77.6	71.7	67.7	76.2
Potatoes	82.4	79.1	86.0	86.6	82.0	91.8
Other	58.1	60.4	56.1	52.3	51.5	53.2
Soybean	90.2	92.8	87.8	92.2	91.9	92.5
Orchards	78.6	68.3	92.4	79.4	75.0	84.3

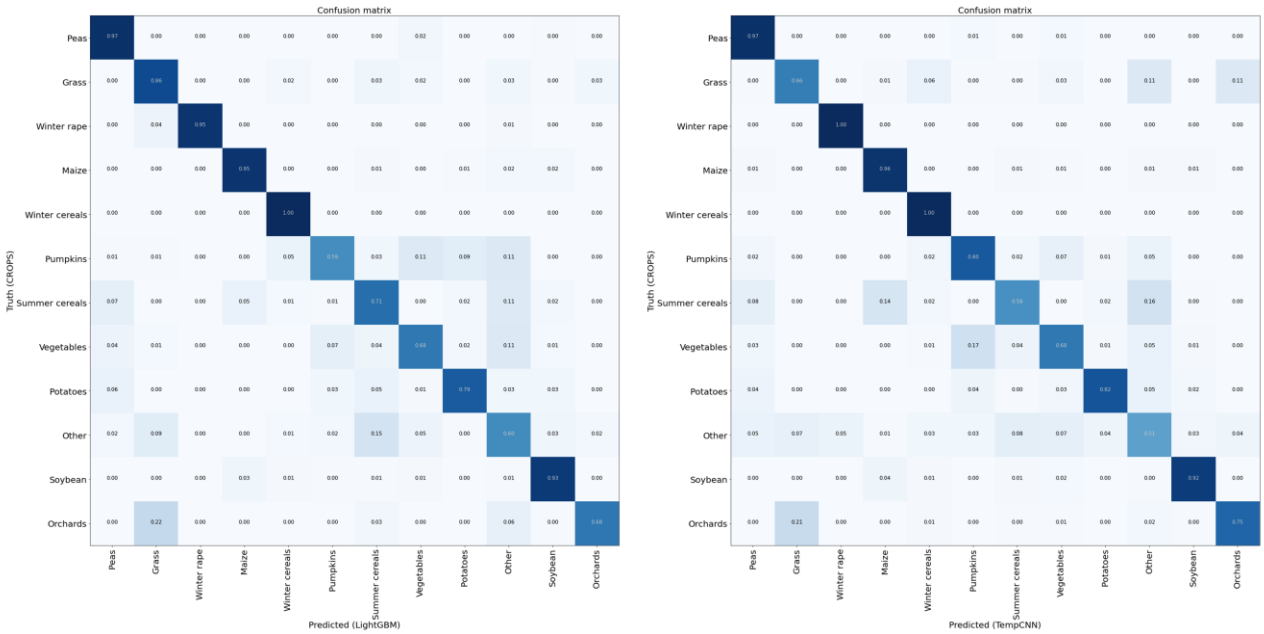

Figure 17: Confusion matrices for random forest (left) and TCNN (right) approach.

The validation of the models shows that for most of the groups both perform very well, but with differences in their confusion for certain classes. The final step in the example is visualisation of the results, as we can see in Figure 18 (shown for RF approach).

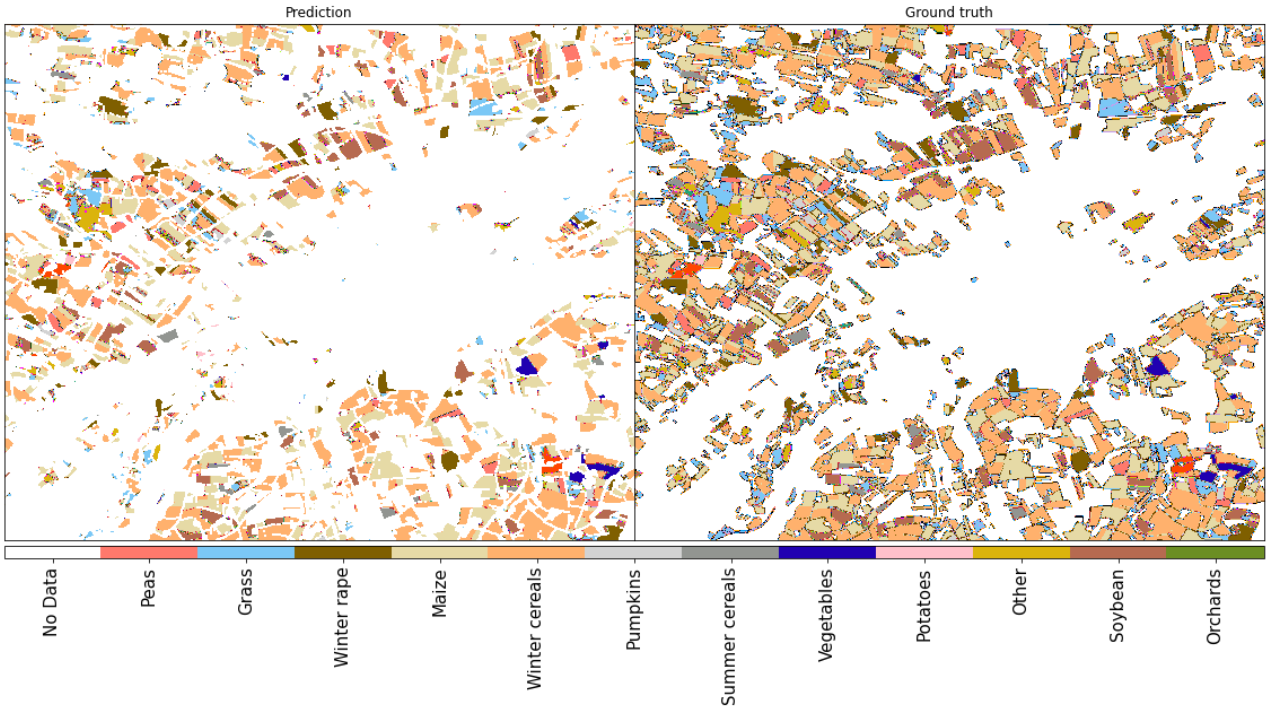

Figure 18: Comparison of RF prediction (left) and ground truth (right) data.

The notebook example goes deeper in trying to understand the differences between the two approaches, what are the reasons for better or worse performance over particular crop type class, and gives some insights on how to address them. It also shows the complexities and computational efficiency of the two approaches. Overall, it serves as a starting point for users to try this approach on their ground truth data, and allows them to improve on the limitations found.

4.2.4 Super Resolution using Sentinel-2 and Deimos imagery example

The temporal resolution of Sentinel-2 imagery allows users to focus on the time-series aspect of the data while its spatial resolution is sufficient for the majority of use-cases. Sometimes, though, ground sampling distance (GSD) of 10 meters just isn't enough. One can resort to commercially available very high resolution (VHR) imagery, but monitoring large areas over longer time intervals with such imagery is very costly. This is where super-resolution (SR) techniques can help. Although VHR imagery is still needed to train the model, the amount of required VHR data is significantly reduced. While the resulting images are not as good as native high-resolution images, they can still provide an approximation that can bridge the gap between cost and quality.

The blog post describes the approach to super-resolve Sentinel-2 bands from their native 10 m to a 2.5 m GSD, therefore a 4x resolution enhancement. The research was done within the DIONE H2020 project where one of the missions is using novel techniques to improve the capabilities of satellite technology while integrating various data sources.

The approach uses `eo-learn` for retrieving and pre-processing both Sentinel-2 and Deimos VHR imagery to the point when the test and training datasets are constructed. The model is adapted from ElementAI's HighResNet architecture, shown in Figure 19, which was modified to fit the specific data preparation pipeline. In this use-case, the problem was framed as a Multi-Frame Super Resolution problem, where noisy low-resolution images of a scene are combined to reconstruct a high-resolution version.

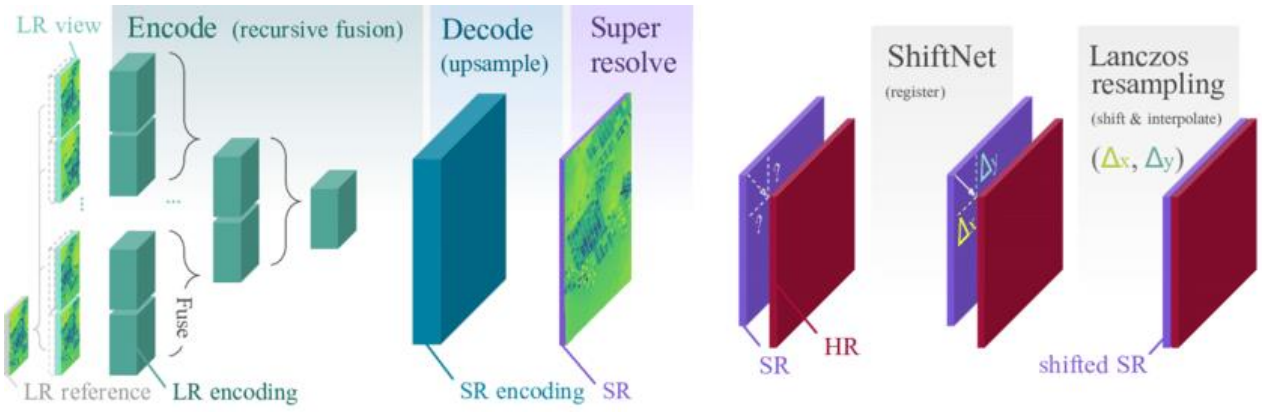

Figure 19: HighResNet architecture, image from reference paper.

The comparison of Sentinel-2 image and MFSR image is shown in Figure 20. From a qualitative point-of-view we can observed that:

- the sharpness increase can be particularly noticed for high contrast features like roads, but less so for low-contrast features like agricultural land;
- the model is able to learn texture, e.g. of trees in forest. Unfortunately, it also learns to predict shadows due to low sun elevation angles, which are more prominent in Deimos imagery since the images were acquired on average 2 hours before Sentinel-2 images. The predicted shadows can be noticed when looking at taller structures like buildings, or trees;
- the model takes as input a temporal sequence of cloudless Sentinel-2 images (i.e. 8 frames) that can span over multiple weeks. However, the model predicts super-resolved images that contain features from the latest Sentinel-2 image. This means that a super-resolved image for each cloudless Sentinel-2 image can be predicted in a rolling-window fashion.

Figure 20: Comparison of Sentinel-2 and super-resolved image.

The published code will hopefully encourage anyone to try the workflow and make it work on their data and area of interest. While this use-case was tailored to Deimos imagery, which is not publicly available, any other

VHR imagery can be used with some adjustments to the workflow. The code makes use of the PyTorch framework, and the implemented architecture, methods for calculating metrics and loss functions will be moved into the interface between `eo-learn` and PyTorch in the coming months.

5 Conclusions

In this deliverable we have shown the integration of `eo-learn` with popular ML and DL frameworks. In particular, the integration with conventional ML frameworks is already in place, and has been used extensively. From the numerous DL frameworks, we have decided to interface `eo-learn` with two: TensorFlow and PyTorch. In order to avoid dependency, maintenance and implementation issues, we have decided that the gateways – interfaces between `eo-learn` and DL frameworks – will be developed as standalone packages. The integration with `eo-learn` and implementation of a number of DL models using TensorFlow framework has already been released in the `eo-flow` package. In the continuation of the GEM project, PyTorch interface will be added in a similar fashion, extending `eo-learn` workflows to another large DL community.

In the main part of the deliverable, Section 4, we give a short description of a number of examples, using `eo-learn` and either conventional or deep learning approaches, together with links where to find them. We hope they serve as good starting points that users can take and adapt to their use-cases.

With the gateways to ML and DL methods in place, we will focus on development of new algorithms and models, focused particularly to support GEM use-cases. Additional examples will be added to the libraries to provide guidance to future users of `eo-learn` on how to implement ML algorithms in their workflows and processes.