This project has received funding from the European Union's Horizon 2020 innovation action programme under grant agreement No 870373 – SnapEarth.

Project 870373

H2020-SPACE-2018-2020

DT-SPACE-01-EO-2018-2020

Deliverable D3.2
Title: Earth Signature software v1

Dissemination Level:	*PU*
Nature of the Deliverable:	*OTHER*
Date:	**26/10/2021**
Distribution:	*WP 3*
Editors:	**QWANT**
Reviewers:	**CERTH, CSR**
Contributors:	**QWANT, METU**

Abstract: This document presents the development of the SnapEarth EarthSignature service. The EarthSignature service is one of the central components of the SnapEarth project. The service extracts the semantic information from the Sentinel2 images then makes them available via a public API. This document describes our progress in the development of this service and which features remain to complete it.

**** Dissemination Level:*** *PU= Public, RE= Restricted to a group specified by the Consortium, PP= Restricted to other program participants (including the Commission services), CO= Confidential, only for members of the Consortium (including the Commission services)*

***** Nature of the Deliverable:*** *R= Report, DEM= Demonstrator, Pilot, Prototype, DEC= Websites, patent fillings, videos, etc., OTHER= Other, ETHICS= Ethics requirement, ORDP= Open Research Data Pilot, DATA= datasets, microdata, etc.*

Disclaimer

Revision History

Date	Rev.	Description	Partner
31/03/2021	01.0	Document creation	Qwant
31/05/2021	01.1	Update with the description of the training process	METU
31/07/2021	01.2	Update with comments from the review	Qwant
01/10/2021	01.3	Updates with comments from the internal review	Qwant, METU
18/10/2021	01.4	Various template updates	QWANT
25/10/2021	01.5	Updates with comments from the internal review	METU, Qwant

List of Authors

Partner	Author
Qwant	Hicham Randrianarivo
Qwant	Hubert Condemi
METU	Gökberk Cinbiş
METU	Samet Çetin

List of Reviewers

Partner	Author
CERTH	A. Zamichos
CERTH	M. Tsourma
CERTH	T. Efthymiadis
CSR	C. Udroiu

Glossary

CLC	Corine Land Cover

Table des matières

Executive summary

This document presents the development of the SnapEarth EarthSignature service to be developed in the WP3 of the SnapEarth project.

EarthSignature is the service that provides the interpretation of Sentinel2 products for all the pilots in the project. The goal of EarthSignature is to extract land cover information from the raw pixels of a Sentinel2 product. We achieve this task by performing semantic segmentation methods to assign a category to each product pixel. The classes we define for this task are critical because they represent the utility of our service. It is the reason we decided to use the categories from the Corine Land Cover (CLC) taxonomy for our primary segmentation model. Still, we also provide models for the pilots with specifics needs.

The document will present the architecture of EarthSignature and the progress so far in its implementation. We will describe how we get the products, the processes we applied to them, how we train the models, and how we make the segmentation results available for the pilots.

1. Introduction

1.1 Scope of the deliverable

This document reports the status and achievements of Task 3.3, EarthSignature software v1. This deliverable aims to present EarthSignature’s architecture and describe its implemented components and those that are not yet implemented. The deliverable is composed of this document and an appendix describing the messages exchanged between EarthSignature and the services that will query its database. The appendix also describes the endpoints to connect to the service.

The document is structured as follows. Section 1 provides an overview of the EarthSignature service, starting with a brief description of the document and an overview of its architecture. Section 2 describes the status of each service at the date of delivery. The section is divided into three subsections corresponding to one module of the service. For each subsection, we provide an overview of the architecture of the components. We describe the role of each element, and finally, we describe the technologies we used for the development.

1.2 Architecture overview

EarthSignature is a service whose goal is to download periodically new Sentinel 2 products, interpret theirs content using semantic segmentation prediction then store the result in a database. This task is performed weekly until the v2 of the service, then will be done daily for the final version of the service. The pilots will access the service using a gRPC API[1]. Figure 1 is a coarse view of the architecture, and in each of the next subsections, we present a detailed picture of the architecture of the service.

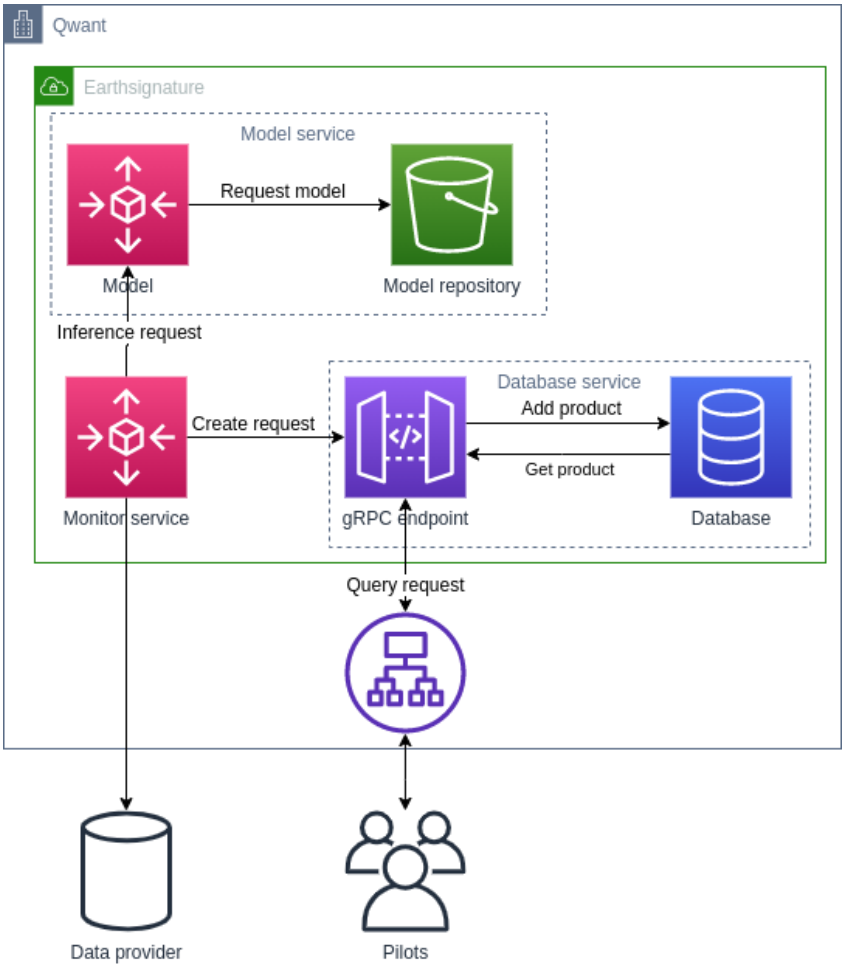

Figure 1: Architecture overview of the EarthSignature service

In the project, we focus on products that cover the European territory. The EarthSignature service contains three components:

- **The Monitor service** which is the component that handles the products download and processing of the data. It first searches the available products for a given date on a provider and then downloads

[1] https://grpc.io/docs/what-is-grpc/

them. Then the service prepares the products for inference using the Model service and finally sends them to the Database service. During the first period of the project, the service run weekly and at the end of the project it should run daily.

- **The Model service** which is used to apply a specific model to a processed product. The service also manages the entries and the updates of the models. The service must handle at least three segmentation models:
    - A model for the CORINE Land Cover
    - A model for burnt areas
    - A model for fine-grained segmentation of crops.
- **The Database service** which is the component that stores all the information generated by the model and downloaded from the provider. The service is publicly accessible via a gRPC API to query a list of inferred products. This service is also internally accessible within EarthSignature so that new entries can be added to the database.

The pilots have access to the segmented product using the database gRPC API.

2. Services Implementation status

2.1 Monitor service

The Monitor service is the heart of EarthSignature. It is the service that checks if new data is available from the provider. It downloads and pre-processes them for the Model service and finally format and send the data to the Database Service. We show the whole process in Figure 2.

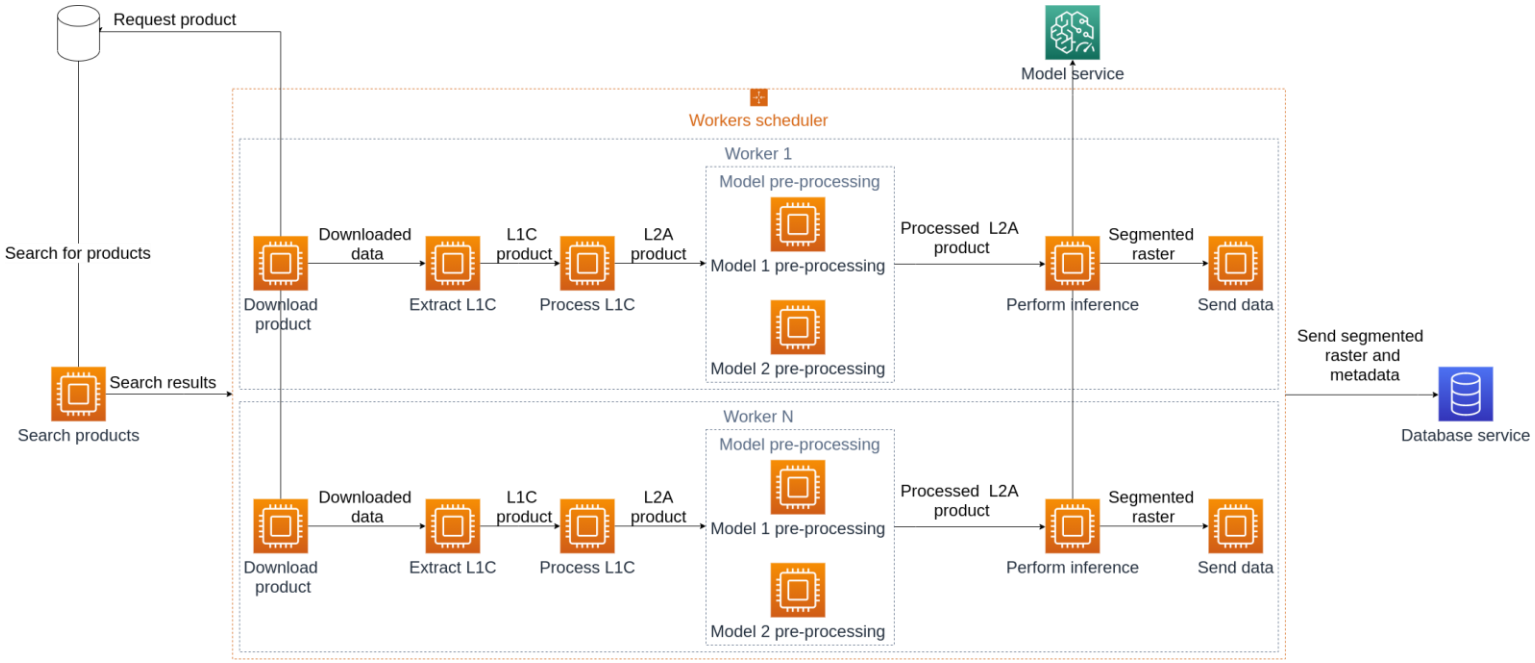

Figure 2: Architecture overview of the Monitor service

This figure shows the different processing applied to an image before sending it to the database. The service first performs a search request to the provider to retrieve the new products of the current day. The search results are then sent to a scheduler that schedules a worker for each product in the search's results. The workers are independent and can be run in parallel. Finally, each worker sends its output to the Database service using the gRPC API.

The downloader module can download the product from various providers. This module uses the EODAG library to download the products from the providers, a list of the available provider can be found in the EODAG documentation[2].

The segmentation model expects L2A products as input. However, this kind of product is not available from every provider. In contrast, L1C products are available from most providers, so we download L1C products by default. In the next version of the service, we will download by default L2A products when available to save computing time.

[2] https://eodag.readthedocs.io/en/latest/getting_started_guide/providers.html.

The product processor is the module that transforms L1C products to L2A products using the Sen2Cor library from the ESA. Also, machine learning models usually need to standardize input data and apply model-specific transformations called preprocessing. The product processor also performs the preprocessing of the data like the normalization of the images or the re-arrangements of the channels.

Luigi[3] is the task manager we use to handle the whole pipeline. Luigi is a package that helps you build complex pipelines of batch jobs. It takes dependency resolution, workflow management, visualization, handling failures, command line integration, and much more. Figure 3 shows the dashboard of the Luigi scheduler. It summarizes the status of the tasks in the pipeline. Using this dashboard, we can monitor the status of the different tasks and particularly which jobs are running, which tasks are finished, and which tasks failed.

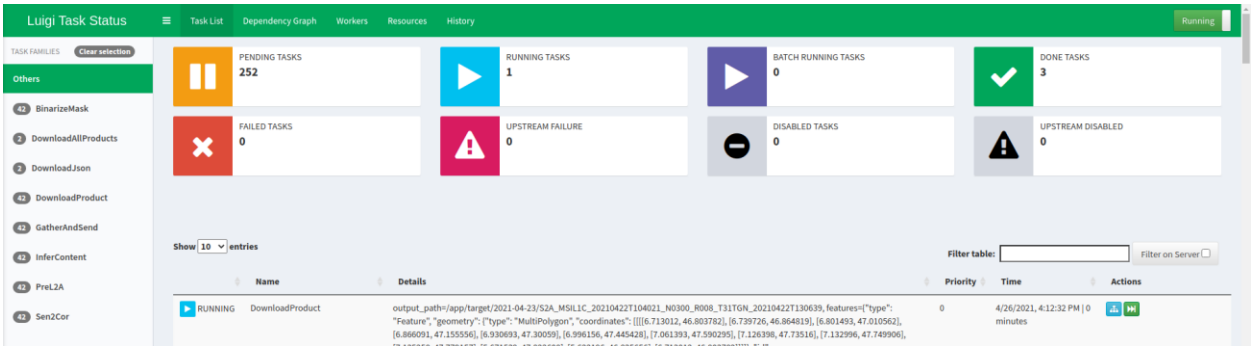

Figure 3: Overview of the dashboard of the Luigi software

Luigi schedules the tasks of the pipeline as a directed acyclic graph (DAG). Each dot in Figure 4 represents a task scheduled by Luigi. The tasks run following the link between them. Using the dependencies between the tasks, Luigi can efficiently run them in parallel. Each line corresponds to the different processing that will be performed on each image. The color of each dot corresponds to the status of the task. If the dot is red, the job failed if the dot is green, the job succeeded, if the dot is yellow, the job is waiting to be run, and if the dot is blue, the task is still running.

[3] https://luigi.readthedocs.io/en/stable/

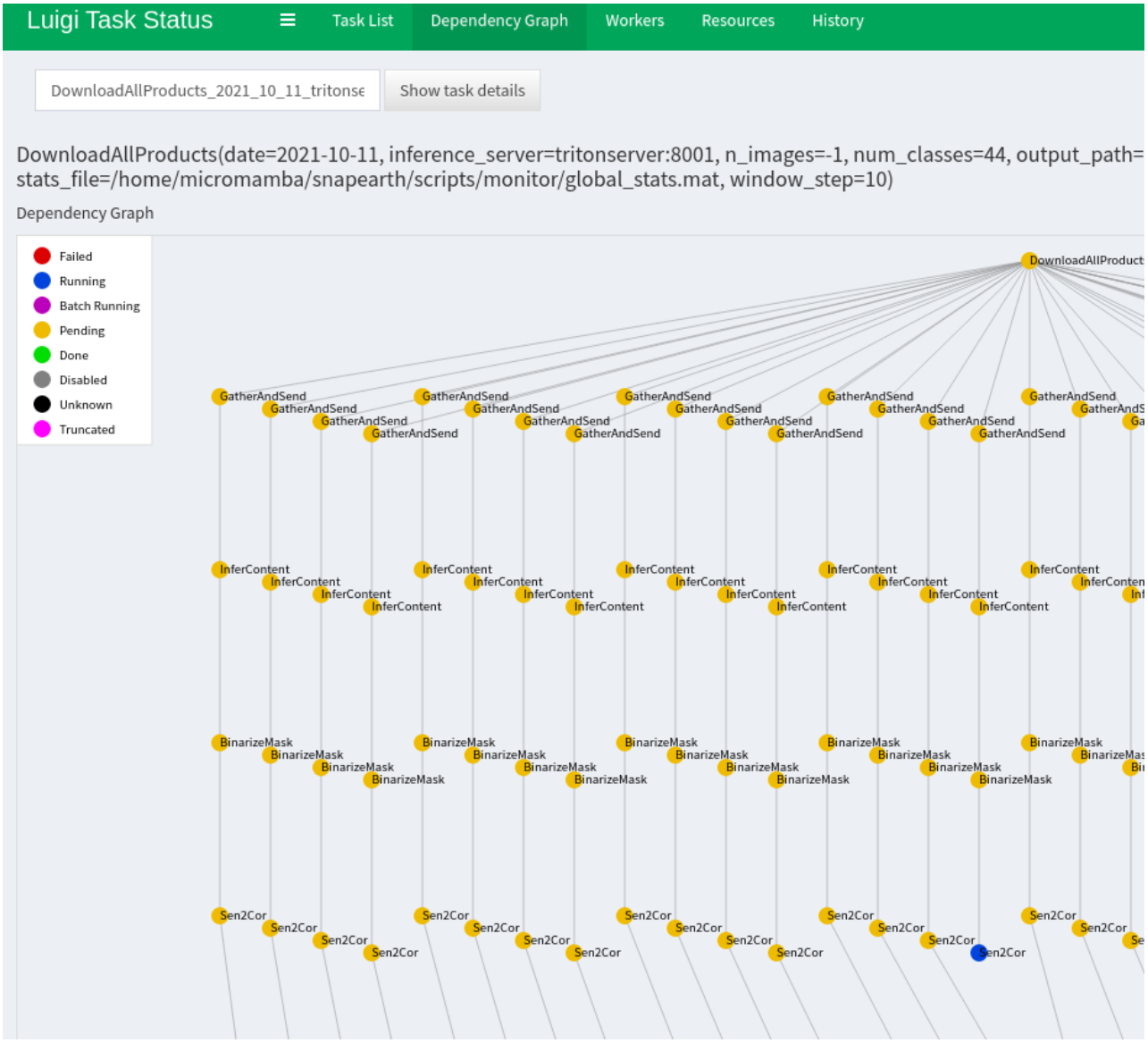

Figure 4: Visualization of the data processing pipeline

Table 1 gives the status of the features that must be implemented in the Monitor Service. The majority of the features planned for the service are implemented. The remaining features include the integration of the models trained on custom datasets.

Table 1 Status of the modules in the Monitor service

Module	Status
Downloader	Finished - Download L1C product - Unarchive result - Store resulting products
Product processor	In progress - Transform L1C to L2A product: finished - CORINE Land Cover model preprocessing: finished - Burt Areas preprocessing: not started

2.2 Model service

To provide a high-quality segmentation service, EarthSignature uses a deep-learning-based model. Serving deep neural networks can be challenging, so we developed this service to serve the various models developed in the project efficiently. Figure 5, shows an overview of the architecture of the service. A client sends input data formatted for the inference. This data will be scheduled for inference, and if there is enough data, it will be batched to improve latency. The server also manages the lifetime of a model. It loads it when needed and manages resources for the model.

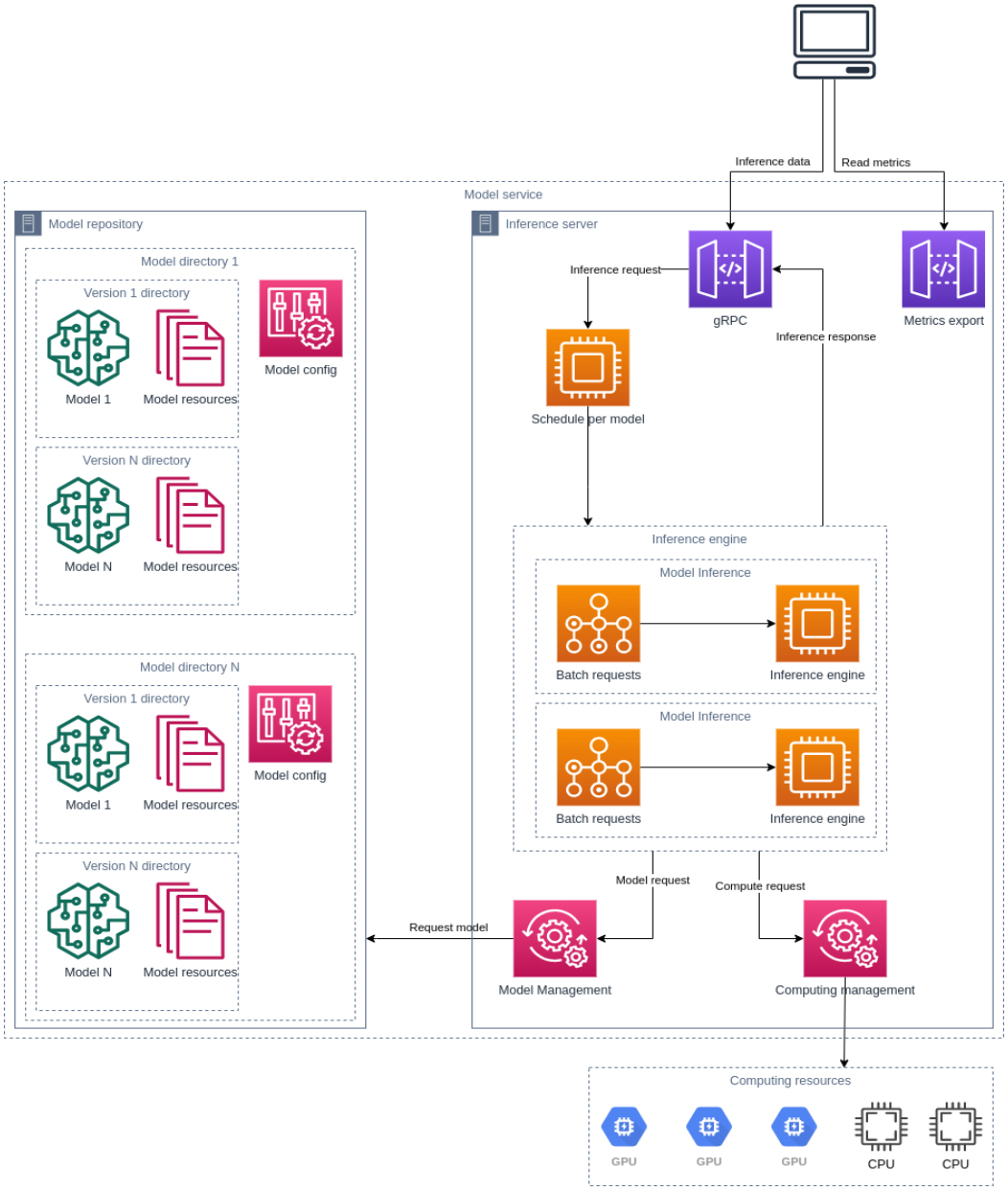

Figure 5: Architecture overview of the Model service

The model service performs two tasks. First, it manages the different models by keeping track of their history and making them accessible for inference using the model repository component. Second, it serves the models to the client using the Inference server using appropriated computing resources. Thanks to this component, we centralize deployment, updates, and access to the models of EarthSignature for easier access to the clients.

The following section describes the process for training the model and which data are used for the training.

2.2.1 Training process used in EarthSignature Software V1.

This section presents the deep learning approach used in constructing the models for EarthSignature Software V1. In particular, the section defines and explains the problem, overall approach, network architecture, training pipeline, and experimental setup details.

Problem definition: EarthSignature uses a deep learning based semantic segmentation model to provide a high-quality service. The semantic segmentation model maps pixels of an input image or a satellite product to one of predefined land cover categories. An important challenge in training is that the annotation masks have much lower resolution (100m) than Sentinel-2 satellite imageries which have 10m/px and 20m/px spatial resolution. In addition, visual inspection shows that the CLC annotations can occasionally be incorrect in larger regions as well. Therefore, the CLC annotations provide weak and incorrect labels for both training and validation purposes (as shown in Figure 6), yielding a challenging weakly and noisily supervised learning problem.

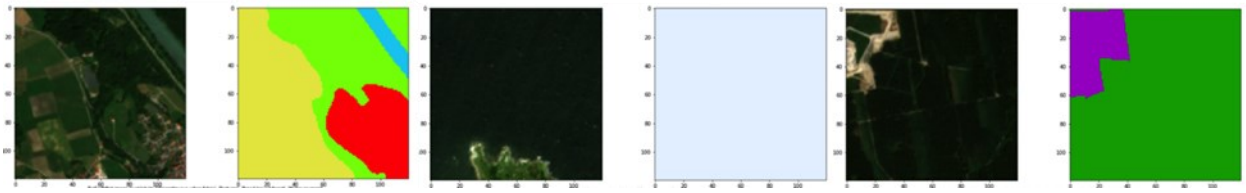

Figure 6: Examples of noisy CLC samples Left: Image, Right: (noisy) CLC annotation.

Main approach: Deep convolutional neural networks, in particular Convolutional Neural Networks (CNNs), are used in the construction of the semantic segmentation model. The model takes a preprocessed input image (described below) and passes through the deep convolutional layers, and outputs a prediction tensor. The prediction tensor contains the probability distribution over target classes for each pixel of the input image. By applying a stochastic gradient descent-based optimization algorithm and minimizing the pixel-wise categorical cross-entropy loss between ground-truth and predicted annotation masks in an iterative manner, the network is trained towards predicting per-pixel true land cover classes.

Architecture: The semantic segmentation model is a modified version of a single scale, fully-convolutional UNet semantic segmentation network [1], illustrated in Figure 7. The model consists of a contracting path and an expansive path, which gives it the U-shaped architecture. The contracting pathway is a typical convolutional network that consists of repeated application of convolutions with 3x3 kernels, each followed by a batch normalization and a rectified linear unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions followed by repeated application of convolutions with 3x3 kernels each followed by a batch normalization and a ReLU, and concatenations with high-resolution features from the contracting pathway. At the end of the network architecture, there is a single convolution operation with 1x1 kernel which maps the number of feature channels to a number of predefined land cover categories followed by a soft-max activation function during training.

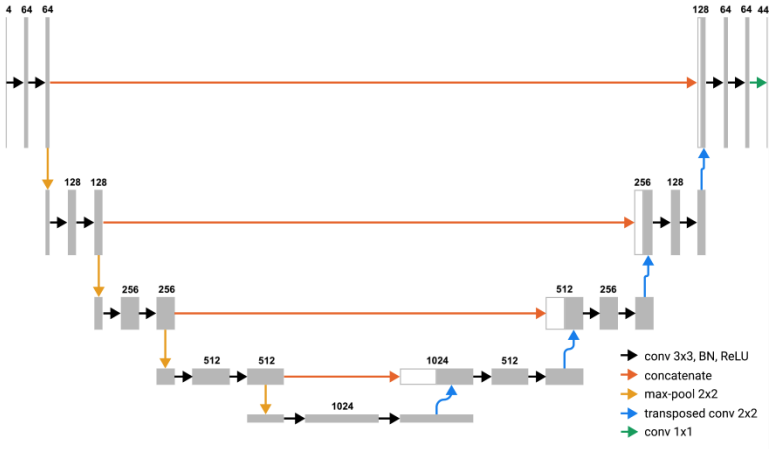

Figure 7: Architecture of the modified Unet semantic segmentation model

Each gray box represents the feature maps in each convolutional layer. The number of channels used is indicated on top of the gray boxes (Adapted from https://innolitics.com/articles/medical-image-segmentation-overview/).

Training pipeline. The training process of the model is divided into several stages. The first stage in the training pipeline applies the necessary preprocessing on input images and annotation masks. For the images, several training data statistics (max, min, channel-wise-mean) are computed and input images are min-max scaled and channel-wise mean-centered, in order. Then, left-right flipping, up-down flipping, and random rotation in the interval of [-180°,+180°] are applied on both images and annotation masks, each with a 50% probability as augmentation methods.

The final semantic segmentation model is trained for 34 epochs with a batch size of 128 and a learning rate of 1e-3 with an Adam optimizer with beta coefficients of 0,9 and 0,999. The train and validation loss plots

can be found in Figure 8 (the dataset details are provided below). At epoch 34, validation loss takes its lowest value.

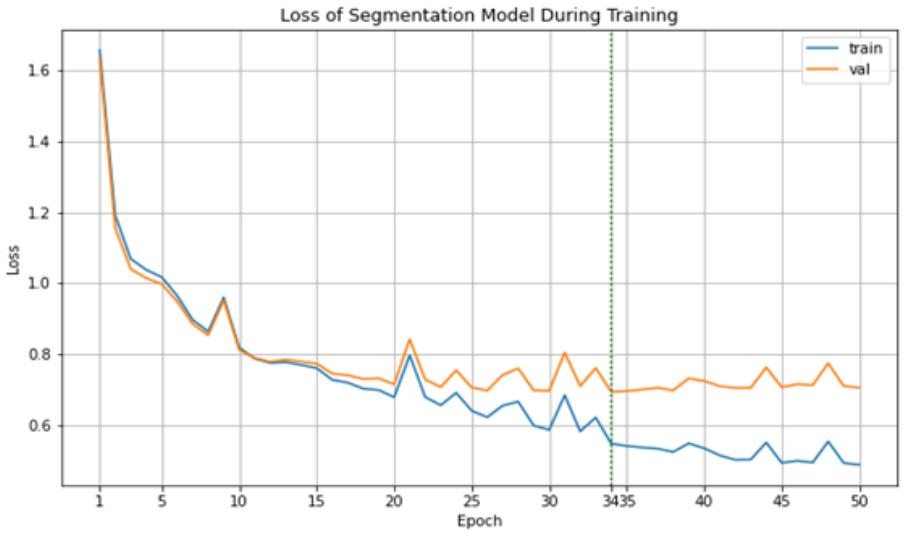

Figure 8: Loss plot of the semantic segmentation model during training

In addition to the main model, pilots (or external users) may need to train problem-specific models with classes outside the CLC Level 2 Nomenclature. For such cases, pilot-specific datasets provided by the partners are used for the construction of pilot-customized versions of land cover segmentation models. For this purpose, the general-purpose model is adapted to these ad-hoc needs through model fine-tuning methodologies. In particular, the layers of the pre-trained deep semantic segmentation network are frozen, the final classification layer is replaced with a randomly initialized layer that performs classification among only pilot-specific target classes. Currently, only the classification layer is adapted to the target due to data limitations, however, the approach can easily be extended to full-fledged fine-tuning in the presence of a large-scale pilot-specific data set.

Experimental setup: The main semantic segmentation model is developed using three different datasets. The CLC dataset contains noisy annotations from Corine Land Cover 2018[4] and is used for training and testing. HRL[5] and URBAN-ATLAS[6] based datasets are more fine-grained and less noisy in their annotation quality when compared to the CLC dataset and are used for evaluation only. In addition, a burnt-area only dataset, provided by CERTH (therefore, referred to as the CERTH dataset) is built with satellite imageries and corresponding burnt area annotations and is used to evaluate the performance of pilot-specific training.

For the development and evaluation of the main model, a dataset with 515131 120x120 images of Sentinel-2 L2A products is constructed. Each image has four spectral bands in 10m resolution (Band-02, Band-03, Band-04, Band-08) and their corresponding Corine Land Cover (CLC) 2018 pixel-level annotation masks. Image crops are taken from BigEarthNet[7] dataset and corresponding pixel-level annotation masks are extracted in an automated way from the CLC 2018 data. The dataset is divided into three splits (training,

[4] https://land.copernicus.eu/pan-european/corine-land-cover/clc2018
[5] https://land.copernicus.eu/pan-european/high-resolution-layers
[6] https://land.copernicus.eu/local/urban-atlas
[7] http://bigearth.net

validation, and test splits) by following the proposed splits of the BigEarthNet paper. There are 267362, 122844, and 124923 samples in training, validation, and test splits, respectively. The semantic segmentation model is trained with samples from the training split and its performance is monitored on validation split to tune hyper-parameters and to avoid overfitting. The final performance of the model is reported using samples belonging to the test split.

The constructed HRL and URBAN-ATLAS benchmarks contain 4 and 19 classes, which are mapped to their counterparts in CLC Level 3 nomenclature. Therefore, a CLC-trained model can directly be evaluated on HRL and URBAN-ATLAS benchmarks. These datasets are divided into three splits in the same way as the splitting procedure as the CLC dataset. However, since the HRL and URBAN-ATLAS datasets are only used for validation, their training splits are ignored and model performance on crops belonging to validation and test splits are considered as the final test performance. The samples belonging to the CERTH dataset are also divided into three splits and samples belonging to train split are used for pilot-specific fine-tuning, samples belonging to validation split are used for monitoring, and samples belonging to test split are used to evaluate the final performance of the fine-tuned model for the pilot-specific task.

Constructions of pilot-specific models: Pilots (or external users) may need to train problem-specific models with classes outside the CLC Level 2 Nomenclature. For such cases, pilot-specific datasets provided by the partners are used for the construction of pilot-customized versions of land cover segmentation models. For this purpose, the general-purpose model is adapted to these ad-hoc needs through model fine-tuning methodologies. In particular, the layers of the pre-trained deep semantic segmentation network are frozen, the final classification layer is replaced with a randomly initialized layer that performs classification among only pilot-specific target classes. Currently, only the classification layer is adapted to the target due to data limitations, however, the approach can easily be extended to full-fledged fine-tuning in the presence of a large-scale pilot-specific data set.

2.2.2 Status of the model (V1)

Evaluation metrics: There are two accuracy metrics being used to compute the performance of the model on the aforementioned datasets. *Average normalized accuracy* is a simple metric that calculates the ratio of the sum of true positive and true negative classifications overall classifications. *Intersection Over Union (IoU)* quantifies the percent overlap between the target mask and predicted mask in segmentation problems and when a false prediction occurs, it does not only penalize the class of which ground-truth label of the pixel belongs, also penalizes the class of which predicted label belongs. Therefore, it is a stricter metric compared to the more commonly used average normalized accuracy. The IoU score is calculated for each class separately and then averaged over all classes to provide a global mean IoU score of semantic segmentation prediction.

The main experimental results are presented in Table 2. From the results, it can be observed that IoU is indeed a much stricter metric. In addition, we observe that while the CLC evaluations are noisy (due to label noise and weak annotations), the model preserves comparable performance on HRL and URBAN-ATLAS datasets. We also notice that for the binary segmentation problem on the CERTH dataset, the model yields high accuracy scores, showing that the model has the potential to be successfully adapted to novel tasks.

Table 2 Average normalized accuracy and intersection over union scores of the segmentation model on test sets of CLC, HRL, URBAN-ATLAS and CERTH datasets

Dataset	Accuracy *(Test)*	
	Average Normalized Accuracy	**Intersection Over Union (IoU)**
CLC (43 classes)	61.4%	48.3%
HRL (4 classes)	71.5%	60.3%
URBAN-ATLAS (19 classes)	62.3%	31.1%
CERTH (2 classes)	80.6%	73.4%

Complementary qualitative results can be found in Figure 9, Figure 10, Figure 11, and Figure 12 CLC, HRL, URBAN-ATLAS, and CERTH datasets, respectively. The CLC result in Figure 9 suggests that the model is able to predict large regions correctly. However, the details differ in CLC ground truth vs model predictions. It should be noted that neither the model nor the ground truth is fully correct, as the ground truth misses important fine-grained details, some of which seem to be recovered by the model. This case shows that with the weak and noisy annotations provided by CLC, it is not only challenging to train a segmentation model, but it is also difficult to validate and evaluate it.

Figure 10 and Figure 11 present example prediction results on the HRL and URBAN-ATLAS benchmarks, which provide finer-grained ground truth annotations. These results suggest the V1 model is able to predict most of the large regions correctly, with mispredictions in small details. These shortcomings are most likely caused by the weak and noisy training annotations during training.

Finally, Figure 12 provides two example predictions using the CERTH model and benchmark. The result shows that the model is able to produce detailed and overall accurate predictions thanks to the accurate training data.

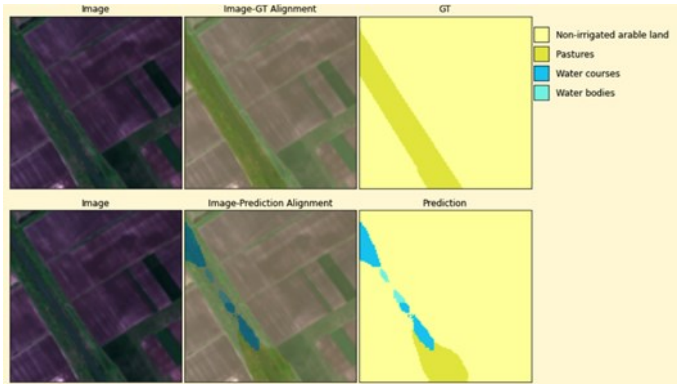

Figure 9: Visualization of a validation sample from CLC dataset. Left-Column: Input Image, Right-Top: (Noisy) Ground-Truth Annotation, Right-Bottom: Prediction of the Segmentation.

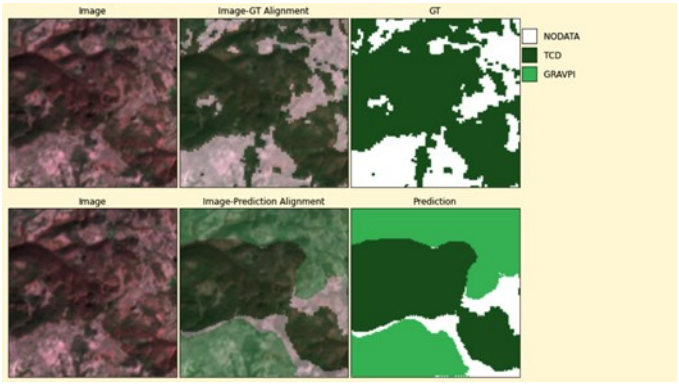

Figure 10: Visualization of a validation sample from HRL dataset. Left-Column: Input Image, Right-Top: Ground-Truth HRL Annotation, Right-Bottom: Prediction of the Segmentation Model.

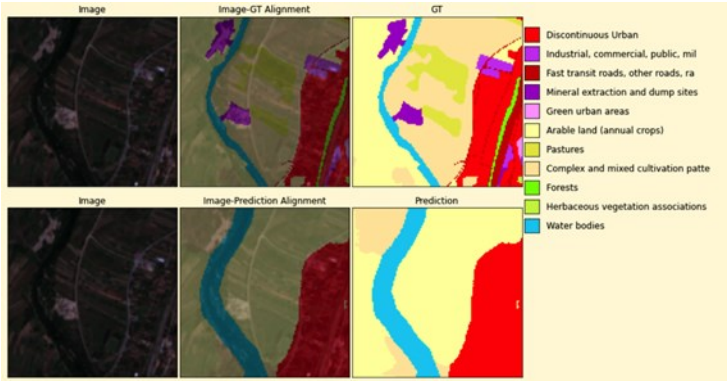

Figure 11: Visualization of a validation sample from URBAN-ATLAS dataset. Left-Column: Input Image, Right-Top: Ground-Truth URBAN-ATLAS Annotation, Right-Bottom: Prediction of the Segmentation Model.

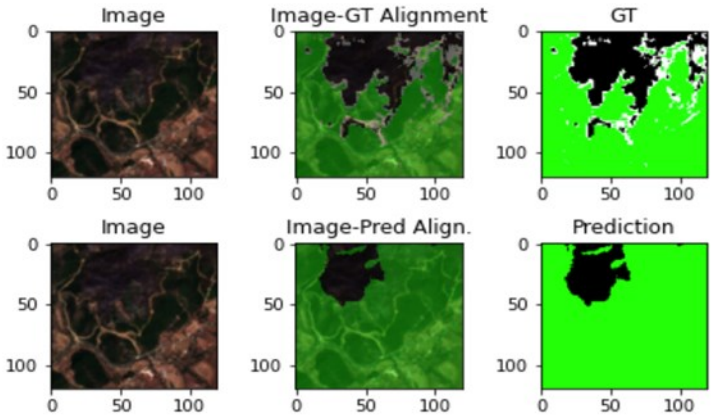

Figure 12: Visualization of a validation sample from CERTH dataset. Left-Column: Input Image, Right-Top: Ground-Truth CERTH Annotation, Right-Bottom: Prediction of the Segmentation Model.

Comparison to the literature: Our goal in V1 has been to establish a strong baseline for follow-up work (for V2) to work on more advanced for handling weakly supervised data. V2 report is planned to contain additional comparisons with the incorporation of updated approach(es). Since there were no benchmarks that satisfy the requirements for EarthSignature development, the aforementioned benchmarks were defined and realized as part of the work in the project, and, the approach has been engineered in a principled manner using these benchmarks. However, still, a rough comparison to an existing recent work can be desirable.

To this end, we have found one independent work [2] which is the only comparable work that we are aware of, similarly studies deep semantic segmentation over the weak and noisy labels of CLC annotations, in combination with Sentinel-2 imagery. It is neither possible to fully reproduce the benchmark of this dataset due to missing details, nor of direct interest for EarthSignature development goals. However, due to the similarity of the CLC benchmark formulated in T3.3 and the experimental setup of this manuscript, a general comparison can be made even if the setups are not fully compatible.

The work by Ulmas & Liiv [2] studies CLC-based semantic segmentation using a maximum class granularity of 25 classes, based on CLC Level 3. According to the confusion matrix reported in the manuscript, the proposed approach yields a normalized accuracy of 21%. Using 15 classes based on the CLC Level 2 nomenclature, the manuscript reports an average normalized accuracy of 31%. In contrast, the model used in EarthSignature V1, as reported above, yields a normalized accuracy of 61.4% over 43 classes. We note that the segmentation problem, naturally, becomes more difficult as the number of classes increases. Therefore, despite the roughness of the comparison, the clear superiority of the results achieved in EarthSignature highlights the strength of the state of the V1 model.

Summary: The models used as part of the EarthSignature Software V1 provide land cover segmentation results that are ready for exploitation. The overall deep learning approach is planned to be improved towards better serving the user (in particular, the SnapEarth pilots) needs.

2.2.3 Efficient model serving

Once the model is available, it can be challenging to serve it efficiently and update it without impacting the rest of the system. Most Machine Learning practitioners embed their model in an HTTP server and query it using a REST API. Following the best practices for serving a model at a large scale, in the SnapEarth project, we choose to use a dedicated server developed by NVIDIA Triton Inference Server[8]. Triton Inference Server simplifies the deployment of AI models at scale in production. It is an open-source inference-serving software that lets teams deploy trained AI models from any framework from local storage or remote storage on any GPU- or CPU-based infrastructure. As the models are retrained continuously with new data, the developers can easily update models without restarting the inference server without disrupting the application.

NVIDIA Triton server comes with a gRPC client library we can use to submit inference requests. However, the functionality of this client is low-level and not very user-friendly. We develop a wrapper around this client to query the server more efficiently for the project's specific needs.

Table 3 gives the status of the features that must be implemented in the Model Service. All the features provided by this service are implemented

Table 3 Status of the modules in the Model service

Module	**Status**
Inference server	Finished: - Server deployment - Models configuration
Inference gRPC client	Finished
Model repository	Finished: - Model repository for specifications - Repository deployment
Segmentation model	version 20201118

2.3 Database service

The database service is responsible for storing and serving the segmentation map extracted from the products. This service relies on a Postgresql[9] database with the PostGIS[10] extension to store geographical information about the satellite products. The pilots can directly query this service using the gRPC protocol. Figure 13 shows an overview of the architecture of the database API. There are two routes when a client

[8] https://developer.nvidia.com/nvidia-triton-inference-server
[9] https://www.postgresql.org/
[10] https://postgis.net/

queries the API. The first is available internally to the project to create a product in the database, and the second is public and allows retrieving the information from the database.

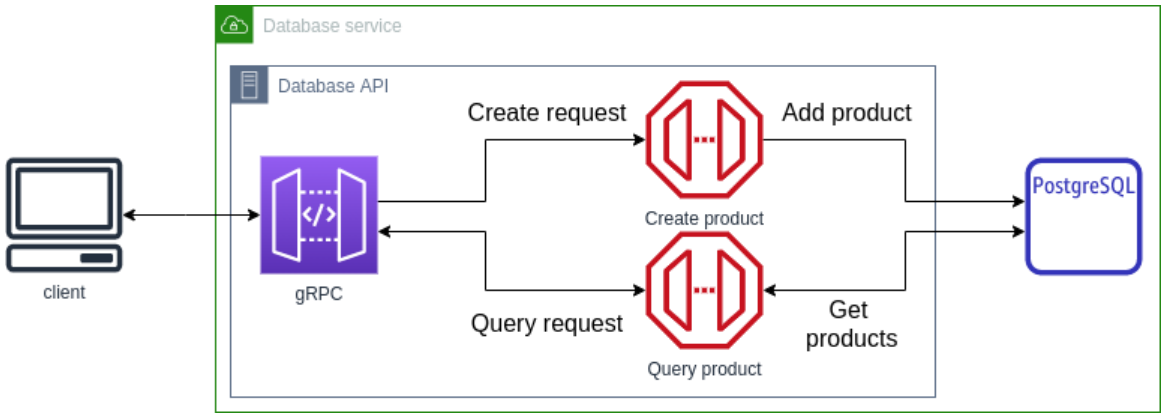

Figure 13: Architecture overview of the Database service

This service is the component that will be available online. It creates new entries in the database using the data sends by the Monitor service and allows pilots to perform a query to the database in a secure way. Communication between the pilots and the service is performed using the gRPC protocol. The format of the messages and the description of the endpoint is detailed in the Appendix. Figure 14 describes the schema of the database.The database contains one entry per product at each date. For each product processed, there is one segmentation maps by category that corresponds to the prediction returned by the model service.

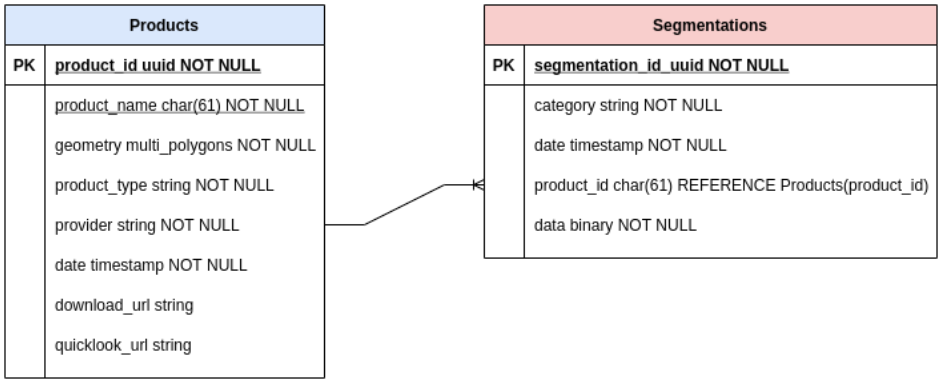

Figure 14: Schema of the EarthSignature database

The pilots will have access to the service definition file that will allow them to implement their client. In the appendix, we provide detailed documentation of the messages and endpoints of the service.

The endpoint for querying the service is available at earthsignature.snapearth.eu:443

Table 4 gives the status of the features that must be implemented in the Database Service. The remaining features for this service are the authentication of the users and modifying the format of the exchanged message with the client to reduce the bandwidth used by an API call.

Table 4 Status of the modules in the Database service

Module	**Status**
Database specification	Finished: - Design database schema - Deploy database server - Initialisation script
Database gRPC server API	Finished: - Add product endpoint: finished - Query product by geometry: finished - Query product by date: finished In Progress: - Query product by categories: in progress - Use GeoJSON/Geobuf format as returned format: in progress
Database gRPC client	Finished
Authentication	Finished: - Mutual authentication using TLS certificates In Progress: - Protocol to exchange certificates

3. Conclusion

In this document, we presented the various services that compose the EarthSignature software for version 1. It comprises three main components, the Monitor service, the Database service, and the Model service. We present an overall architecture schema that describes how the services are connected and the messages exchanged. The complete description of these messages is in the Appendix with the list of the endpoints provided by EarthSignature.

We also provide an extensive description of the models we integrate into this service. We describe the datasets used, our training protocol, and how we evaluated the models. We explain how the models are integrated with the service using the Model Service.

For the next period, for model training, we plan to work on updating the current weakly supervised learning strategy by incorporating limited amount of auxiliary high-resolution / non-noisy pixel level annotations. We plan to develop a new approach for this purpose, report its evaluation results with comparisons to the related approaches.

For the service, we plan to reduce the time spent processing an image without increasing the computing capacity of the service. We will use multiple recent methods for model inference published by manufacturers like Intel or Nvidia to achieve this goal. We also want to reduce the size of the data stored for each product. Today we stock them as binary large objects, and we will benchmark if storing them as vector data may be more efficient for our use case.

Finally, we enumerate the progress of the EarthSignature Software by listing the progress of the implementation of the features in each module.

References

[1] Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

[2] Ulmas, P., & Liiv, I. (2020). Segmentation of satellite imagery using u-net models for land cover classification. *arXiv preprint arXiv:2003.02899*.

Appendix

Protocol Documentation

Table of Contents

snapearth/api/v1/segmentation.proto

Top

Raster

A representation of the segmented image. The data field corresponds to the prediction of a pixel in the original product.

The prediction can be one of the 44 classes of the CORINE Land cover or it can be 0-1 to denote the presence or not of cloud.

Field	Type	Label	Description
height	int32		Height of the product
width	int32		Width of the product
data	uint32	repeated	List of caterories in the image

snapearth/api/v1/database.proto

Top

CreateProductRequest

The request message to create a new product in the database

Field	Type	Label	Description
data	geobufproto.Data		Geojson describing the product
segmentation	Raster		segmentted product
cloud_mask	Raster	optional	Cloud mask. 1 mean cloud 0 no clouds

CreateProductResponse

Empty message

ListSegmentationRequest

Message used to query the database. All the fields are optional but there is a limit to the number of results returned

Field	Type	Label	Description
wkt	string	optional	area to query in Well Know Text format. Optional if product_ids is set
start_date	google.protobuf.Timestamp	optional	product start acquisition date
end_date	google.protobuf.Timestamp	optional	product end acquisition date
product_ids	string	repeated	Products id to filter
categories	string	repeated	Categories to filter

n_results	int32	optional	

SearchSegmentationRequest

Field	Type	Label	Description
query	string		Query string for the search
n_results	int32	optional	
descriptor_set	google.protobuf.FileDescriptorSet		

SegmentationResponse

Field	Type	Label	Description
wkt	string		area of acquisition of the product
segmentation	Raster		Raster array where each pixel is mapped to a category
cloud_mask	Raster	optional	Cloud mask. 1 mean cloud 0 no clouds
product_id	string		
quicklook	string		TODO add timestamp TODO add geotransform

DatabaseProductService

Service that exposes the EarthSignature database.

The service exposes two endpoints for its consumers

- ListSegmentation to query a list of segmentation responses

- CreateProduct to add a new product in the database

Method Name	Request Type	Response Type	Description
ListSegmentation	ListSegmentationRequest	SegmentationResponse stream	Query the database and returns a list of SegmentationResponse
CreateProduct	CreateProductRequest	CreateProductResponse	Add a new product in the database
SearchSegmentation	SearchSegmentationRequest	SegmentationResponse stream	Method to search a list of segmentation with textual query

Methods with HTTP bindings

Method Name	Method	Pattern	Body
ListSegmentation	GET	/v1/segmentation:list	*
SearchSegmentation	GET	/v1/segmentation:search	*

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHP	Ruby
double		double	double	float	float64	double	float	Float
float		float	float	float	float32	float	float	Float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/string	Bignum
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer	Bignum or Fixnum (as required)
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/string	Bignum or Fixnum (as required)
sint32	Uses variable-length encoding. Signed int value. These more efficiently	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)

	encode negative numbers than regular int32s.							
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/string	Bignum
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer	Bignum or Fixnum (as required)
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/string	Bignum
sfixed32	Always four bytes.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/string	Bignum
bool		bool	boolean	boolean	bool	bool	boolean	TrueClass/FalseClass
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string	String (UTF-8)
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string	String (ASCII-8BIT)

Protocol Documentation

Table of Contents

geobufproto/geobuf.proto

Top

Data

Field	Type	Label	Description
keys	string	repeated	global arrays of unique keys
dimensions	uint32	optional	max coordinate dimensions Default: 2
precision	uint32	optional	number of digits after decimal point for coordinates Default: 6
feature_collection	Data.FeatureCollection	optional	
feature	Data.Feature	optional	
geometry	Data.Geometry	optional	

Data.Feature

Field	Type	Label	Description
geometry	Data.Geometry	required	
id	string	optional	
int_id	sint64	optional	
values	Data.Value	repeated	unique values
properties	uint32	repeated	pairs of key/value indexes
custom_properties	uint32	repeated	arbitrary properties

Data.FeatureCollection

Field	Type	Label	Description
features	Data.Feature	repeated	
values	Data.Value	repeated	
custom_properties	uint32	repeated	

Data.Geometry

Field	Type	Label	Description
type	Data.Geometry.Type	required	
lengths	uint32	repeated	coordinate structure in lengths
coords	sint64	repeated	delta-encoded integer values
geometries	Data.Geometry	repeated	
values	Data.Value	repeated	
custom_properties	uint32	repeated	

Data.Value

Field	Type	Label	Description
string_value	string	optional	
double_value	double	optional	
pos_int_value	uint64	optional	
neg_int_value	uint64	optional	
bool_value	bool	optional	
json_value	string	optional	

Data.Geometry.Type

Name	Number	Description
POINT	0	
MULTIPOINT	1	
LINESTRING	2	
MULTILINESTRING	3	
POLYGON	4	
MULTIPOLYGON	5	
GEOMETRYCOLLECTION	6	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHP	Ruby
double		double	double	float	float64	double	float	Float
float		float	float	float	float32	float	float	Float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/string	Bignum
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer	Bignum or Fixnum (as required)
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/string	Bignum or Fixnum (as required)
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/string	Bignum
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer	Bignum or Fixnum (as required)
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/string	Bignum
sfixed32	Always four bytes.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/string	Bignum
bool		bool	boolean	boolean	bool	bool	boolean	TrueClass/FalseClass
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string	String (UTF-8)
	May contain any arbitrary							

bytes	sequence of bytes.	string	ByteString	str	[]byte	ByteString	string	String (ASCII-8BIT)