OGC Testbed-14

Machine Learning Engineering Report

Table of Contents

Publication Date: 2019-02-04

Approval Date: 2018-12-13

Submission Date: 2018-06-14

Reference number of this document: OGC 18-038r1

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D030

Category: Public Engineering Report

Editor: Tom Landry

Title: OGC Testbed-14: Machine Learning Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

This OGC Engineering Report (ER) describes the application and use of OGC Web Services (OWS) for integrating Machine Learning (ML), Deep Learning (DL) and Artificial Intelligence (AI) in the OGC Testbed-14 Modeling, Portrayal, and Quality of Service (MoPoQ) Thread. This report is intended to present a holistic approach on how to support and integrate emerging AI and ML tools using OWS, as well as publishing their input and outputs. This approach should seek efficiency and effectiveness of knowledge sharing.

This engineering report will describe: experiences, lessons learned, best practices for workflows, service interaction patterns, application schemas, and use of controlled vocabularies. It is expected that the description of workflows for geospatial feature extraction will be more complex than the implementations found in the deliverables.

1.1. Requirements & Research Motivation

The AI landscape is rapidly evolving, leading to new possibilities for geospatial analytics on ever larger Big Data. Current standards are being modernized to take into account these novel methods and the new data available in federated infrastructures. The goal of this work is to develop a holistic understanding of AI, ML, and DL in the context of geospatial. By this work, it is expected to advance or derive best practices for integrating ML, DL, and AI tools and principles in the context of OWS. The following figure presents an overview of the research motivations to be addressed.

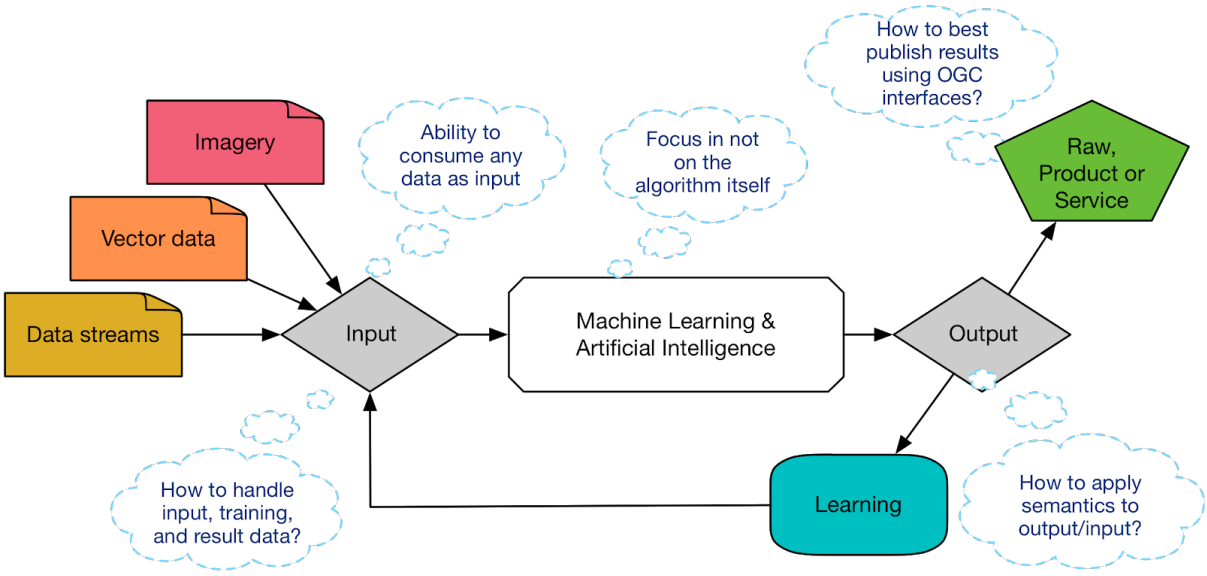

Figure 1. Overview of the ML task.

Throughout their experiments, the participants of this task needed to answer the following questions:

- What are the main interface requirements?
- What are the main interactions between components?
- Does the proof of concept allow for support of all geospatial datatypes, including vector information?
- Are there interfaces or requirements specific to gridded imagery to introduce?

- Is the proof-of-concept independent of algorithm type?

Several approaches were proposed for consideration in this work:

- Consider training data handling and performance information of the ML tools
- Compare "move-the-algorithm-to-the-data" and "move-the-data-to-the-algorithm"
- Describe transparency and accountability principles
- Describe testing of ML algorithms on fully-scaled infrastructure

1.1.1. Additional requirements

Two additional requirements were to be addressed in this work:

- Data shall be made available at OGC data access services e.g. Web Feature Service (WFS), Web Coverage Service (WCS), Sensor Observation Service (SOS)
- Use of catalog services is optional, but desirable

1.2. Prior-After Comparison

1.2.1. Main findings

Previous work where OGC prototyped a Web Image Classification Service (WICS) interface (OGC 05-017) [1] that defined *GetClassification*, *TrainClassifier* and *DescribeClassifier* operations. WICS provided support for unsupervised classification, supervised classification using trained classifiers, and supervised classification using classifiers trained based on client-supplied training data. In this work, the D165 Machine Learning (ML) system describe these operations: *TrainML*, *RetrainML* and *ExecuteML*. The ML Knowledge Base (KB) describes the *GetModels*, *GetImages*, *GetFeatures*, and *GetMetadata* operations, as well as their *Store* counterparts. The Controlled Vocabulary Manager (CVM) offered by the D166 Semantic Enablement of ML task enables searching of terms by specifying *namespace*, *prefix*, and other parameters using *facet* searches.

The work described in this ER improves on WICS [1] by presenting additional findings on one ML classifier and one Deep Learning (DL) classifier. Additionally, this work presents findings from another DL classifier mainly used to transfer learning from EO workflows onto pretrained DL models. Details on these Additional ML systems can be found in the Approaches and good practices section. Therefore, the use of three different ML systems in this task allowed experimentation that further enabled definition of interoperable standards supporting ML, DL, and, by extension, AI.

The disaster concept considered in the Proof of Concept (POC) section has potential impacts and implications for numerous information communities such as Geospatial Intelligence (GeoINT) and Earth systems science. This work is therefore expected to support the OGC Disasters Interoperability Concept Study [http://www.opengeospatial.org/projects/initiatives/disasterscds] conducted simultaneously with Testbed-14. Initial concepts discussed very early in the ML task did consider the use of Internet-of-Things (IoT) sensor data, but was dismissed from implementations due to complexity. The POC demonstrates the use of both Very High Resolution (VHR) and Synthetic Aperture Radar (SAR) imagery, offering distinct spatiotemporal characteristics and enabling different applications. Section 5 offers a quick survey of the AI, ML, and DL landscape and briefly

presents applications and challenges for GeoInt and Earth Systems. The reader can also refer to the bibliography for an overview of the literature considered.

1.2.2. How to best support ML and AI using OWS?

This ER notes the following uses of OWS that enabled support of ML in the testbed:

- Implementation of a Representational State Transfer (REST) and JavaScript Object Notation (JSON) binding for a transactional Web Processing Service (WPS) to improve interoperability with Exploitation Platform experiments in the Earth Observation & Clouds (EOC) thread
- Develop ML systems following Application Packaging best practices as advanced in EOC and as found in [2]
- Develop an experimental WPS, Web Map Service (WMS), and Web Feature Service (WFS) for model interoperability and transparency, such as described in discussion section 9.6.
- Use of a Web Map Tile Service (WMTS) and OGC 17-041 Vector Tiles [http://docs.opengeospatial.org/per/17-041.html] as input for ML annotation, training, and inference operations, as described in the discussion in Section 9.8.

1.2.3. How to best publish input to and outputs from ML and AI using OWS?

This ER notes the following uses of OWS to better publish input and outputs of ML:

- Use of an OGC Catalogue Service for the Web (CSW) as an interface for the Knowledge Base (KB), as described in the discussion in Section 9.2.
- Use of an implementation of the OGC ISO Application Profile of CSW to handle information in KB
- Use of CSW for data exchange and as an interface of the Controlled Vocabulary Manager (CVM), as described in the discussion in Section 9.7.
- Consider use of OGC WFS 3.0 to manage annotated and output features, and more generally as a reference service interaction pattern
- Consider use of ISO 19115 and/or OGC CSW-ebRIM Application Profile
- Use of the OGC® Open Modelling Interface (OpenMI) Interface Standard [https://www.opengeospatial.org/standards/openmi] for advanced description of models

1.3. Recommendations for Future Work

A goal of this task and its analysis was also to suggest potential future activity where these results could be investigated through recommended future tasks, deliverables, components, and Engineering Reports. This section presents recommended potential future tasks and deliverables that can support advancement of requirements and expand research motivations. The reader can also refer to Section 9 this Engineering Report for discussion and open issues for further details.

1.3.1. Recommended Future Tasks

1. Advance temporal enablement of ML through experiments with a variety of temporal data at

largely different scales, such as IoT timeseries from sensors, satellite imagery, weather forecasts, climate projections, etc. While selection of data would aim for temporal variety, it would be most likely accompanied by a large spatial variety. Related to Testbed-15 idea: Predictive geospatial analytics through Artificial Intelligence [https://github.com/opengeospatial/ideas/issues/85]

2. Advance Semantic enablement of ML through experiments including data search, data annotation, model training, model interoperability, ML output publishing, metadata, and transfer learning. One goal of this task would be to identify standardization needs and future work for transforming data into knowledge, thus supporting advanced AI use cases.
3. Advance Computational enablement of ML through experiments with a variety of processing methods, infrastructures, and execution environments, including secured workflows, containerization, cloud, and In-Memory MapReduce. One goal of this task would be to ensure that other tasks could design or operate in fully-scaled infrastructures.
4. Provide an OGC Catalogue Service (CSW) interface as a standardised interface into the Knowledge Base and experiment with Next Generation OGC Web services, as described in the discussion on knowledge base.
5. Integrate the CVM into the existing OGC Testbed-14 AI/ML architecture, as described in the discussion on semantic interoperability.

1.3.2. Recommended Future Deliverables

Recommended Future Components

The following components are suggested to be deployed for testing, demonstration, and integration purposes. The resulting functionalities of these components could support the recommended future tasks.

1. ML-enabled EOC application packages and workflows, where a packaged ML system is used in conjunction with EO pre-processing steps, in coherence with Testbed-15 idea: Workflows - Motivating use case [https://github.com/opengeospatial/ideas/issues/75]
2. ML systems that trains models including point clouds as inputs. Related to Testbed-15 idea: Generalizing point cloud data services using OGC standards [https://github.com/opengeospatial/ideas/issues/31]
3. ML systems supporting interfaces for interoperable DL models, where trained models from a ML system can be loaded into a different one to be used for inference. For example, we note ONNX [http://onnx.ai/] for Deep Learning open models, and more generally OGC® Open Modelling Interface (OpenMI) Interface Standard [https://www.opengeospatial.org/standards/openmi] for process simulations
4. Testbed-15 idea: Filter Encoding Standard extension for Imagery [https://github.com/opengeospatial/ideas/issues/88]
5. Transfer learning demonstrators, where a model trained from scratch by one ML system is reused and adapted in a second ML system, possibly on a different computational environment
6. Advanced data store that allows efficient and flexible mapping between tiles or other forms of efficient spatiotemporal subsetting and the input layers of Deep Learning architectures

7. Advanced knowledge base and search capabilities that allow better transparency of ML models and support for validation of large numbers of training runs
8. Clients that can support supervised learning at scale such as Testbed-15 idea: Large scale geospatial annotation campaign for Deep Learning applications [https://github.com/opengeospatial/ideas/issues/71]
9. Advanced clients providing tools to interact with all previously mentioned future components, as well as managing user feedback as described by the Geospatial User Feedback (GUF) [https://www.opengeospatial.org/standards/guf] standard
10. Clients and systems that implement security mechanisms such as OAuth 2.0

Recommended Future Engineering Reports (ER)

The following Engineering Reports are suggested to support and document experiments conducted inside previously mentioned recommended future tasks and components.

1. Geospatial ML systems best practices
2. AI for EO Application Packaging and Workflows best practices
3. ML for disaster interoperability

1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization
Tom Landry	CRIM
Cameron Brown	Envitia
Neil Kirk	Envitia
Chih-Wei Kuan	Feng Chia University
Benjamin Pross	52 North
Cullen Rombach	Image Matters
Martin Sotir	CRIM

1.5. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following normative documents are referenced in this document.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2], 2010
- ISO: ISO 19115-2:2009, Geographic information — Metadata — Part 2: Extensions for imagery and gridded data [https://www.iso.org/standard/39229.html], 2009
- OGC: OGC 13-084r2, OGC® I15 (ISO19115 Metadata) Extension Package of CS-W ebRIM Profile 1.0 [https://portal.opengeospatial.org/files/?artifact_id=56905], 2014
- OGC: OGC 07-110r4, OGC® CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW0 [http://portal.opengeospatial.org/files/?artifact_id=31137], 2009
- OGC: OGC 11-014r3, OGC® Open Modelling Interface (OpenMI) Interface Standard [https://portal.opengeospatial.org/files/?artifact_id=59022], 2014
- W3C: A JSON-based Serialization for Linked Data [https://www.w3.org/2018/jsonld-cg-reports/json-ld/], 2018

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- Active learning

Active learning is a special case of semi-supervised machine learning in which a learning algorithm is able to interactively query the user to obtain the desired outputs at new data points. There are situations in which unlabeled data is abundant but manually labeling is expensive. In such a scenario, learning algorithms can actively query the user/teacher for labels.

- Algorithm

An unambiguous specification of how to solve a class of problems. Algorithms can perform calculation, data processing and automated reasoning tasks.

- Annotation

Manual image annotation is the process of manually defining regions in an image and creating a textual description of those regions. Automatic image annotation (also known as automatic image tagging or linguistic indexing) is the process by which a computer system automatically assigns metadata in the form of captioning or keywords to a digital image.

- Application schema

Conceptual schema for data required by one or more applications [SOURCE: ISO 19101-1:2014]

- Artificial intelligence

Artificial Intelligence (AI) is the ability of a computer program or a machine to think and learn. An ideal (perfect) intelligent machine is a flexible agent which perceives its environment and takes actions to maximize its chance of success at some goal. An extreme goal of AI research is to create computer programs that can learn, solve problems, and think logically.

- Artificial neural network

An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. The signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. Artificial neurons typically have a weight that adjusts as learning proceeds. See Neural Network.

- Batch

A subdivision of a dataset into number of batches. A batch is required when the entire dataset is to large to be passed to a neural network.

- Batch size

Total number of training examples present in a single batch.

- Bundle

To bundle software means to sell it together with a computer, or with other hardware or software, as part of a set.

- Class

A class is the category for a classifier which is given by the target.

- Classification

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

- Classification map

A visual representation of the output of classification, often viewed as dense, gridded imagery. See Classifier.

- Classifier

An algorithm or a method that processes to classification of an input. See Classification.

- Cluster

Generally, a group of data objects. Typical cluster models include connectivity models, distribution models, density models and neural models.

- Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other.

- Controlled vocabulary

Controlled vocabularies provide a way to organize knowledge for subsequent retrieval. They are used in subject indexing schemes, subject headings, thesauri, taxonomies and other forms of knowledge organization systems.

- Convolutional neural network

A convolutional neural network (CNN) uses convolutions to extract features from local regions of an input. CNNs have gained popularity particularly through their excellent performance on visual recognition tasks.

- Dataset

Identifiable collection of data.

- Deep learning

Deep Learning is a class of machine learning algorithms that: use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation; learn in supervised and/or unsupervised manners; learn multiple levels of representations that correspond to different levels of abstraction.

- Deep neural network

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. See Artificial Neural Network.

- Detection

Detection includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. The resulting features are subsets of the image domain, often in the form of isolated points, continuous curves or connected regions.

- Entity

A thing with distinct and independent existence. An entity is something that exists as itself, as a subject or as an object, actually or potentially, concretely or abstractly, physically or not. An entity-relationship (ER) model describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types.

- Epoch

Learning iteration on a training dataset.

- Extraction

In machine learning, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction. See Feature.

- Feature

Abstraction of real-world phenomena. A feature can occur as a type or an instance. Feature type or feature instance should be used when only one is meant [SOURCE: ISO 19101-1:2014].

- Feature engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. The need for manual feature engineering can be obviated by automated feature learning. See Feature Learning.

- Feature learning

Feature learning is a set of techniques that allows a system to automatically discover the

representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task. See Feature.

- Fine tuning

Fine-Tuning refers to the technique of initializing a network with parameters from another task, and then updating these parameters based on the task at hand. Fine tuning is a common technique is to train the network on a larger data set from a related domain. Once the network parameters have converged an additional training step is performed using the in-domain data to fine-tune the network weights. This allows convolutional networks to be successfully applied to problems with small training sets.

- Generative adversarial networks

Generative adversarial networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework. One network generates candidates (generative) and the other evaluates them (discriminative). Training the discriminator involves presenting it with samples from the dataset, until it reaches some level of accuracy. See Sample.

- Ground truth

In machine learning, the term "ground truth" refers to the accuracy of the training set's classification for supervised learning techniques.

- Hyperparameter

In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training. Given these hyperparameters, the training algorithm learns the parameters from the data. The time required to train and test a model can depend upon the choice of its hyperparameters. A hyperparameter is usually of continuous or integer type, leading to mixed-type optimization problems.

- Iteration

In machine learning systems, the number of batches needed to complete one epoch.

- Knowledge base

A knowledge base (KB) is a technology used to store complex structured and unstructured information used by a computer system. A knowledge-based system consists of a knowledge-base that represents facts about the world and an inference engine that can reason about those facts and use rules and other forms of logic to deduce new facts or highlight inconsistencies.

- Hidden layer

In traditional feed-forward neural networks, a hidden layer neuron is a neuron whose output is connected to the inputs of other neurons and is therefore not visible as a network output. See Layer.

- Label

See Class.

- Layer

Basic unit of geographic information that may be requested as a map from a server. In the context of neural networks, a layer is an ensemble of neurons in a network that processes a set of inputs and their results.

- Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to progressively improve performance on a specific task with data, without being explicitly programmed.

- Metadata

Data that provides information about other data.

- MLlib

MLlib is Apache Spark's scalable machine learning library. It provides tools such as ML Algorithms, Featurization, Pipeline, Persistence and Utilities.

- Model

A statistical model is a mathematical model that embodies a set of statistical assumptions concerning the generation of some sample data and similar data from a larger population. A statistical model represents, often in considerably idealized form, the data-generating process.

- Neural network

A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

- OpenAPI

The OpenAPI Specification is an Application Programming Interface (API) description format for REST APIs.

- Parameter

In computer programming, a parameter is a special kind of variable, used in a subroutine to refer to one of the pieces of data provided as input to the subroutine. In convolutional layers, the layer's parameters consist of a set of learnable filters or kernels.

- Profile

Set of one or more base standards and - where applicable - the identification of chosen clauses, classes, subsets, options and parameters of those base standards that are necessary for accomplishing a particular function [ISO 19101, ISO 19106].

- PyTorch

PyTorch is an open source machine learning library for Python, based on Torch, used for applications such as natural language processing or image processing. PyTorch provides GPU-accelerated Tensor computation and Deep Neural Networks.

- Remote sensing

Remote sensing is the acquisition of information about an object or phenomenon without making physical contact with the object and thus in contrast to on-site observation. It generally refers to the use of satellite- or aircraft-based sensor technologies to detect and classify objects on Earth, including on the surface and in the atmosphere and oceans, based on propagated signals.

- Segmentation

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments.

- Semantics

A conceptualization of the implied meaning of information that requires words and/or symbols within a usage context.

- Semantic class

A semantic class contains words that share a semantic feature. According to the nature of the noun, they are categorized into different semantic classes. Semantic classes may intersect. See Semantic Feature.

- Semantic feature

Semantic features represent the basic conceptual components of meaning for any lexical item.

- Semantic gap

Separation between the visual content in a digital image and semantic descriptions.

- Semantic interoperability

Semantic interoperability that assures that the content is understood in the same way in both systems, including by those humans interacting with the systems in a given context.

- Semantic segmentation

Semantic segmentation describes the process of associating each pixel of an image with a class label.

- Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

- Synthetic aperture radar

Synthetic aperture radar (SAR) is a form of active remote sensing – the antenna transmits radiation that is reflected from the image area, as opposed to passive sensing, where the reflection is detected from ambient illumination. SAR image acquisition is therefore independent of natural illumination and images can be taken at night. See Remote Sensing.

- Target

Detection of targets is a specific field of study within the general scope of image processing and image understanding. From a sequence of images (usually within the visual or infrared spectral bands), it is desired to recognize a target such as a car.

- Taxonomy

A system or controlled list of values by which to categorize or classify objects.

- Test dataset

A test dataset is a dataset that is independent of the training dataset, but that follows the same probability distribution as the training dataset. The test dataset is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset. A test set is therefore a set of examples used only to assess the performance (i.e. generalization) of a fully specified classifier.

- Tile

A rectangular representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix [source: OGC 07-057r7].

- Training dataset

A training dataset is a dataset of examples used for learning. A model is initially fit on a training dataset, that is a set of examples used to fit the parameters of the model. The model is trained on the training dataset using a supervised learning method.

- Tuning

See Fine Tuning.

- Uncertainty

Two type of uncertainty can be identified, epistemic and aleatoric uncertainty. Epistemic uncertainty captures ignorance about which model generated the collected data. Aleatoric uncertainty relates to information which collected data cannot explain.

- Unsupervised Learning

Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of "unlabeled" data.

- Validation dataset

A validation dataset is a set of examples used to tune the hyperparameters of a classifier. Fitted models are used to predict the responses for the observations in the validation dataset. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters.

- Vocabulary

A language user's knowledge of words. See Controlled Vocabulary.

3.1. Abbreviated terms

- AOI Area of Interest
- AI Artificial Intelligence
- API Application Programming Interface
- CFP Call For Participation
- CRIM Computer Research Institute of Montreal
- CNN Convolutional Neural Network
- COG Cloud Optimized GeoTIFF
- CWL Common Workflow Language
- DGGS Discrete Global Grid Systems
- DL Deep Learning
- DSTL Defense Science and Technology Laboratory
- EOC Earth Observation & Clouds
- ER Engineering Report
- FCU Feng Chia University
- GAN Generative Adversarial Network
- GPU Graphical Processing Unit
- JSON JavaScript Object Notation
- ML Machine Learning
- MGCP Multinational Geospatial Coproduction Program
- MLP Multi Layer Perceptron
- MoPoQ Modeling, Portrayal, and Quality of Service
- NAS NSG Application Schema
- NDAAS NSG Data Analytic Architecture Service
- NGA National Geospatial-Intelligence Agency
- NLP Natural Language Processing
- NLU Natural Language Understanding
- NN Neural Network

- NIEM National Information Exchange Model
- NSG National System for Geospatial Intelligence
- OGC Open Geospatial Consortium
- OSM OpenStreetMap
- OWS OGC Web Services
- RNN Recurrent Neural Network
- SAR Synthetic Aperture Radar
- SatCen European Union Satellite Centre
- SVM Support Vector Machines
- URL Uniform Resource Locator
- USGIF United States Geospatial Intelligence Foundation
- VHR Very High Resolution
- W3C World Wide Web Consortium
- WCS Web Coverage Service
- WFS Web Feature Service
- WMS Web Map Service
- XML Extensible Markup Language

Chapter 4. Overview

Section 5 provides background information found in the state-of-the-art of AI, ML, and DL that is relevant to the ML task.

Section 6 describes the proof of concept developed by the task participants. The concept, scenarios, and use cases are presented. The design of each component, deliverable by deliverable, is described. Implemented systems, processes, and workflows are reported. Metadata and application schemas are described.

Section 7 reports various approaches and good practices as identified by the participants through their own implementations or by references of authoritative material. The section presents best practices of workflows, service interaction patterns, and application schemas.

Section 8 demonstrates the proof of concept. The demonstration scenarios and relevant material are presented.

Section 9 lists the recommendations from the task participants to OGC. Recommendations addressing the creation or extension of OGC standards to support ML and AI geospatial algorithms are presented, as well as recommendations of good practices for the use of tiles or grid structures for the analysis, including DGGS.

Annex A provides an XML WPS 2.0 process description of ExecuteML as presented by the ML system.

Annex B provides an XML WPS 2.0 process description of RetrainML as presented by the ML system.

Annex C provides an XML WPS 2.0 process description of TrainML as presented by the ML system.

Annex D provides a JSON file of the default response of the Controlled Vocabulary Manager.

Annex E provides a JSON file of the full response of the Controlled Vocabulary Manager.

Annex F provides a components table for the ML Task of the MoPoQ thread of Testbed-14.

Annex G illustrates a sample Pleiades VHR image of Paris used to train models and infer classes.

Chapter 5. AI, ML and DL landscape

Section 5 provides background information found in the state-of-the-art of AI, ML, and DL that is relevant to the ML task. This landscape study puts into context key elements from the CFP, supports future work recommendations found in the summary, and lays the groundwork for discussion.

For additional background, the CFP also lists the following sources:

- Big Data DWG: Simple Features for Big Data
- Human-Agent Collectives
- Data Quality work of Testbed-12
- Gal's Thesis on Uncertainty in Deep Learning
- Principles for Algorithmic Transparency and Accountability
- NSG Data Analytic Architecture Service (NDAAS)
- Multinational Geospatial Coproduction Program (MGCP)
- NGA 2020 analysis technology plan

5.1. Background information on ML

Several examples of supervised, unsupervised, and semi-supervised ML applications can be found in [3] and [4]. Typical problem classes are illustrated by [5] in the following figure.

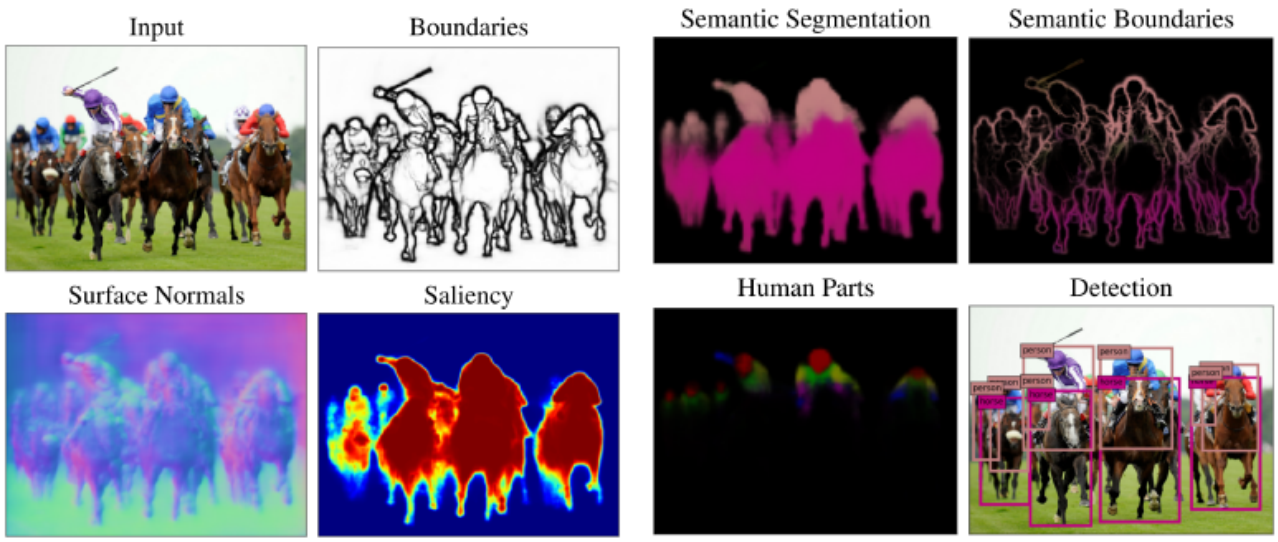

Figure 2. Various tasks performed by Deep Neural Networks (Yuille, 2018)

Major performance metrics of ML systems include *precision* and *recall*, *training time* and *execution time*, and *metaparameters* and *analyst feedback*. Similarly, the CFP lists the following metadata as key elements to consider in the ML task:

- trust levels and confidence
- associated body of evidence
- associated imagery

- optimized learning capabilities
- applied processes
- quality parameters
- links to original images and/or snippets

5.2. GeoINT applications

This section briefly presents key GeoINT challenges coherent with this work's scope, with respect to the role of the analyst, to the processes and workflows, and to scenarios and applications. Further reading material can be found in [6], [7], [8], [9] and [10]. Below is a definition of GeoInt as found on Wikipedia.

```
GEOINT (GEOspatial INTelligence) is intelligence about the human activity on earth
derived from the exploitation and analysis of imagery and geospatial information that
describes, assesses, and visually depicts physical features and geographically
referenced activities on the Earth.
```

5.2.1. Processes and workflows

Several sources often mentioned the use of processes such as metadata tagging, data preparation and ingestion, data search, and filtering. Advanced use cases in Natural Language processing (NLP) findings [11], with systems containing query interpreters and query conductors in federated subsystems, are mentioned. Data challenges presented include hypothesis formulation, confidence tracking, maintaining links to the original data, and transversal of workflows. Similarly, the OGC Testbed-14 CFP lists the following processes and workflows as references for the ML task participants:

- Conversion into a format suitable for feature extraction and classification
- Filtering of images to remove very low quality samples that might skew the ML model
- Provision of metadata to describe the provenance and configuration of the model
- Export of outputs to be re-ingested as inputs for further learning, in a closed feedback loop
- WPS, WFS, and OWS Context Profiles to support development of DL models for geospatial feature extraction

5.2.2. Scenarios and use cases

Several scenarios and use cases of ML can be found in the literature. Usual applications involving detection of objects and events, as found in [9], [12] and [13], include:

- Tracking of human migration
- Finding high value targets such as terrorists
- Locating waste piles that enable breeding of mosquitos
- Monitoring of land cover and determination of land use

Other GeoInt use cases involving forecasting, simulation, and prediction can be found in [8]. In this broad category, we note the following applications:

- Inventory management based on weather patterns
- Regional climate response
- Watershed evaluation
- Agricultural forecasting and food and water security

Finally, in accordance with the disaster scenario described in Section 6, more information can be found in [14], [15], [16] and [9]. For this scenario, we also note the following applications:

- Detecting trends in air pollution
- Flood and inundation models for storm surge prediction
- Simulation of human behavior in catastrophic urban scenarios

5.3. Earth Systems applications

This section briefly presents key Earth Systems challenges coherent with this work's scope with respect to the data characteristics as well as scenarios and applications. Here this engineering report notes a large overlap with GeoInt applications presented above, as Earth Systems studies natural phenomena that impact life.

5.3.1. Data

Data sources often used in Earth Systems include, aerial [17], in-situ, model outputs and remote sensing. From the latter, this engineering report notes the possible estimation of variables [18] such as methane in air, forest cover, global surface water and land cover change (unsupervised). A good overview [19] of the inherent characteristics of gridded data, vector data, and swaths includes:

- Boundaries and spatial dimensionality
- Temporal characteristics and presence of various cycles
- Multiresolution
- Sample and ground truth sizes

Amongst these data sources, the use of the term "features" to describe a transformation or reduction of the input space, for instance in [20]. Rules, heuristics and ad hoc models are often used as handcrafted features [12] [4]. In the context of DL, "deep features" describes the outmost layers of a learned NN, as found in [21] and [22]. Here, this engineering report notes the relevance of work conducted in the Spatial Data on the Web Best Practices WG [OGC 15-107 [https://www.w3.org/TR/sdw-bp/]].

5.3.2. Scenarios and use cases

Several scenarios and use cases of ML can be found in the literature. Usual applications involving detection, tracking, and forecasting of events and phenomena can be found in [18]. In accordance with the disaster scenario described in Section 6, the following applications are also noted:

- hurricanes, tornados, and fires
- weather fronts and atmospheric rivers

To enable and operationalize these Big Data applications, unified systems and databases [19] are created. There is a presence of large-scale ML systems that relies on tiles and DGGS [17], Cloud optimized GeoTIFF (COG), Datacubes and various other spatial databases.

5.4. Annotation datasets

In our scenario, an analyst that must annotate imagery in order to train or retrain models has several modalities offered to him or her. For example, annotation clients and methods found in [4], [17] and [23] use point points, patches, bounding boxes, tiles, contours, and regions. The figure below illustrates the use of polygons to compactly delimitate objects of interest and may be considered as state-of-the-art.

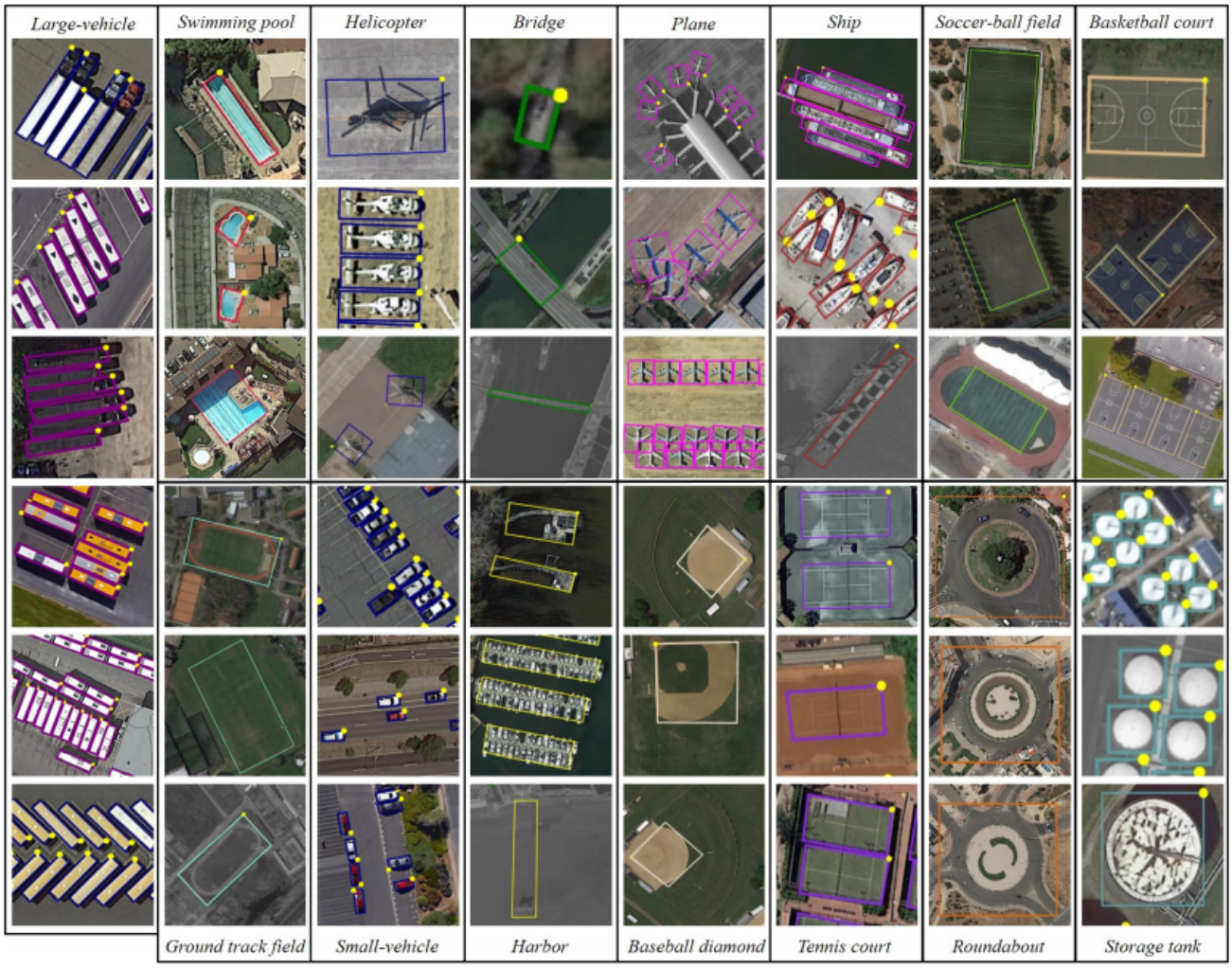

Figure 3. Use of polygons to annotate and locate compactly enclosed objects (Xia, 2018)

Semantic aspects of annotation, including discussions on classes, labels, vocabularies and their granularity, can be found in [21] and [24]. The ML application in [20] describes the use of hierarchies, trees, ontologies, and directed graphs such as WordNet.

A large number of popular authoritative annotated data sources and datasets are referenced in [24], [25], [3], [26], [15], [20], [27] and [28]. These datasets include, but are not limited to, ImageNet,

COCO, LandScan3, Urban 3D Challenge, IARPA, SpaceNet, PASCAL, Semantic Segmentation Challenge, Forest Recognition Competition, Flickr, GoogleImages, and OpenStreetMap. Instructions on how to construct training, validation, and test datasets while taking into account bias and diversity can be found in [17], [5] and [21].

5.5. Deep Learning

This section gives an overview of the main characteristics of DL models and highlights special considerations for geospatial intelligence tasks.

Deep learning refers to the practice of training large, multi-layered, artificial neural networks for supervised or unsupervised ML problems. Since the 90's, neural networks have been known to be a universal function approximator, and stacking several layers was thought to be able to model complex non-linear relations, but training such models remained out of reach. It is not until 2006 that, due to a combination of factors (compute power, suitable datasets, and new training methods), DL models started surpassing the accuracy of other state-of-the-art ML models [29]. Since then, DL has become one of the most active research subjects in the field of ML. It is now at the core of many state-of-the-art models and has yielded many applications in the industry including in the imagery and geospatial data analysis domain.

In comparison with other methods in the field of ML, DL models presents several advantages:

- a simple compute model (direct acyclic graph of linear transform or element-wise non-linearity)
- a relatively straightforward optimization scheme (variants of the stochastic gradient descent)
- potential universality in its application

In practice, DL has shown some limits [5] including:

- difficult to apply on heterogeneous data
- limited interpretability and explainability
- training-set dependent bias
- sensitivity to adversarial examples [30].

Deep learning models excel at processing 1D, 2D, 3D, dense, and homogenous data such as raster images, sound signals, text data, and time series (i.e.: data that can easily be represented as tensors). This effectiveness has led to many breakthroughs in accuracy for image classification, object detection, segmentation, speech recognition, and language modelling tasks. Deep learning-based methods have also yielded some impressive results on generative tasks, in particular with the use of GANs [15] (e.g,: increasing image resolution and generating maps from satellite images).

5.5.1. Special considerations

In the context of the design of ML services for geospatial applications, this section focuses on four aspects: runtime environment, feature engineering, training data availability, and transfer learning.

Special training and runtime environment requirements

Neural network training and inference algorithms are mostly composed of linear algebra operations, convolutions, and element-wise function applications. These operations are highly parallelizable and are well-suited for accelerators like GPUs (Graphical Processing Units), FPGAs (Field-Programmable Gate Arrays), and TPUs (Tensor Processing Units, developed at Google Inc). For instance, DL models for image classification will often run 20 to 50 times faster on a modern GPU than a multi-core CPU. Offloading batched inferences to machines equipped with accelerators may be a consideration when designing ML services for processing large-scale data.

Feature engineering

One of the most attractive aspects of deep neural networks is that they often do not require manual feature extraction from input data. This contrasts with most traditional ML techniques applied to images (e.g., Support Vector Machine) that require extracting relevant patterns from images (e.g., histograms, edges, and key points) beforehand. This is showcased by the most popular DL architecture for image analysis : Convolutional Neural Networks [31], where small convolutions features are automatically learned (inspecting learned filters, one can often identify common edge detection or circular pattern detectors). Having generic, end-to-end models that rely only on annotated data rather than hard-coded feature extraction presents significant advantages for geospatial intelligence tasks. For instance, implementations are more generic, thus encouraging re-utilization.

Labelled training data

Most deep neural networks require a large quantity of labelled training data to make correct predictions for unseen data (generalization). When there are few or no annotated samples, DL methods are often outranked by other ML models. Overcoming this issue is an active research field (few-shot-learning, weakly/unsupervised learning). When designing systems embedding neural networks, it may be relevant to plan measuring and monitoring model generalization performance. Active learning may be used to mitigate cases when data annotation is expensive or time-consuming: based on model generalization performances, a system may automatically suggest samples to be labelled. These samples are chosen to maximize model accuracy increases, keeping the number of manually labelled samples as low as possible.

Transfer learning

One consequence of having end-to-end models capable of solving a large class of problems is that models can often be reused. Trained model parameters can be used to specialize or fine-tune a model. For instance, reusing parameters from a general-purpose image classification model to train a model for fine-grained airliner identification. Model parameters can also be reused to perform different tasks (e.g., from classification to anomaly detection) or to be applied to different data (e.g., from RGB to SAR), this process is described a "transfer learning". Fine-tuning and transfer learning are very common practices that alleviate the lack of annotated data and many applications for earth observation and remote sensing rely on this technique [3] [12]. Having a way to re-use pre-trained models or fine-tune an existing model with new data is an important consideration for ML systems.

Chapter 6. Proof-of-concept

Section 6 describes the proof-of-concept developed by the task participants. The concept, scenarios, and use cases are presented. The design of each component, deliverable by deliverable, is described. Implemented systems, processes, and workflows are reported. Metadata and application schemas are described.

6.1. Concept, scenarios, and use cases

At the onset of the testbed, the concept was refined to include major data sources, systems, and clients. Below is an example of the work produced to prepare the subsequent development efforts and demonstrations.

For the proof-of-concept, the testbed participants decided to focus on disaster interoperability scenarios and, in particular, on urban floods. The disaster management cycle is well-suited for this study, as it is well-documented and encompasses several activities that may involve the use of ML services:

- Prevention and monitoring activities: multiple-source data fusion, water levels monitoring, risk models based on hydrodynamical and meteorological simulations, etc.
- Disaster preparation and response activities: ground operations, situation assessment (e.g., using high resolution images to get updates on the state of infrastructure, damage and human activities), etc.

Those tasks involve multiple data sources (e.g., punctual sensors, ground cameras, high and medium resolution imagery, SAR images) and have high requirements on system interoperability and decision traceability.

Those scenarios are further refined for the demonstration section. In this section, particular scenarios are not described, but rather a background that drives architecture and API design choices. Below is a figure created in the concept generation phase of Testbed-14.

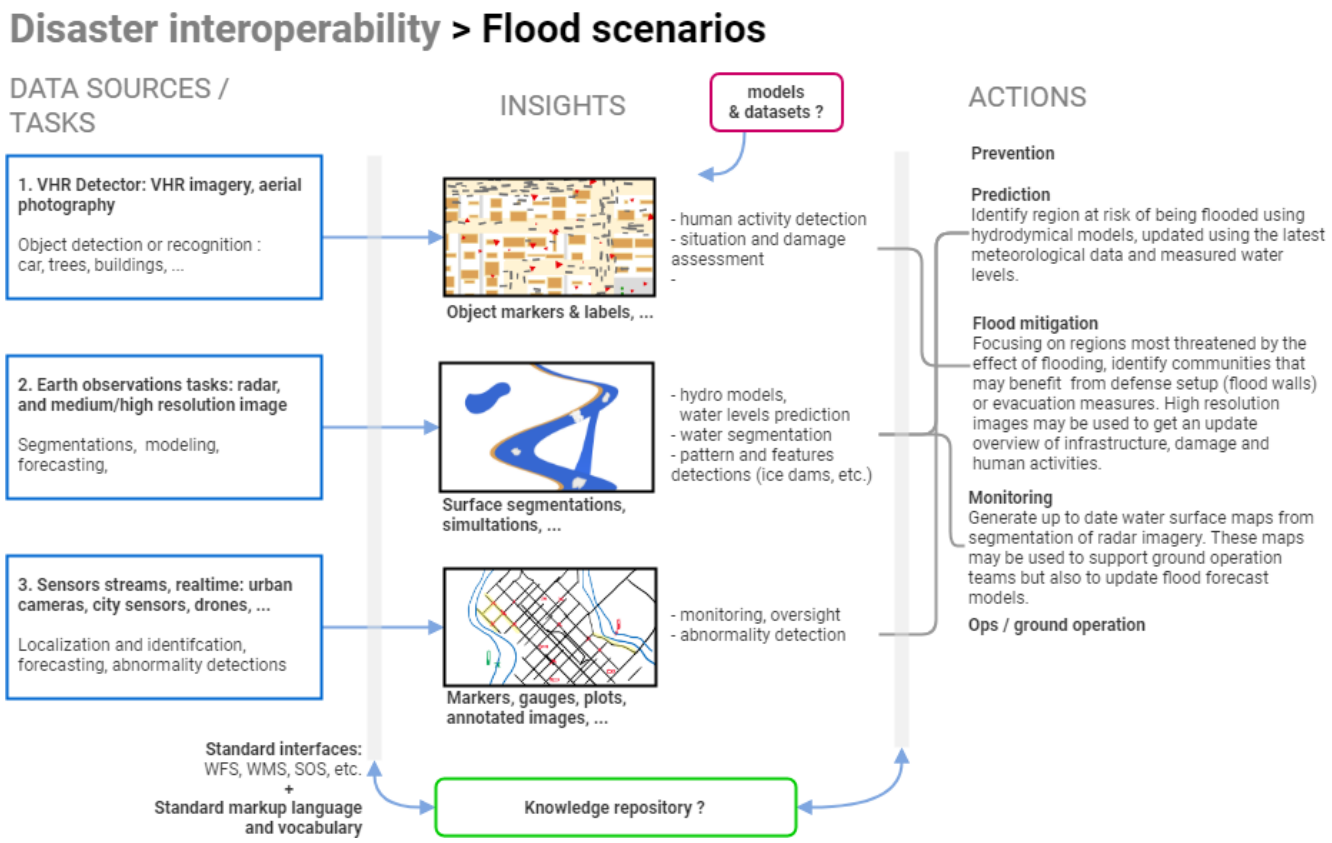

Figure 4. Further refinement of the initial disaster scenario concept.

In the scope the of the ML system, four categories of actors were identified: Image analysts, software engineers, ML model developers, and end users. Theses actors are often split over several public and private organizations, but they all may interact with the same ML system at some point: when building, updating, deploying or using the service APIs.

Their interactions with the system are depicted in the following figure. In the most basic use case, the Image Analyst (IA) uses prior- and post-event data to identify areas destroyed by a natural disaster such as a hurricane. As temporal aspects were put aside in the current work, the image analyst identifies feature types that do not take any change detection into account.

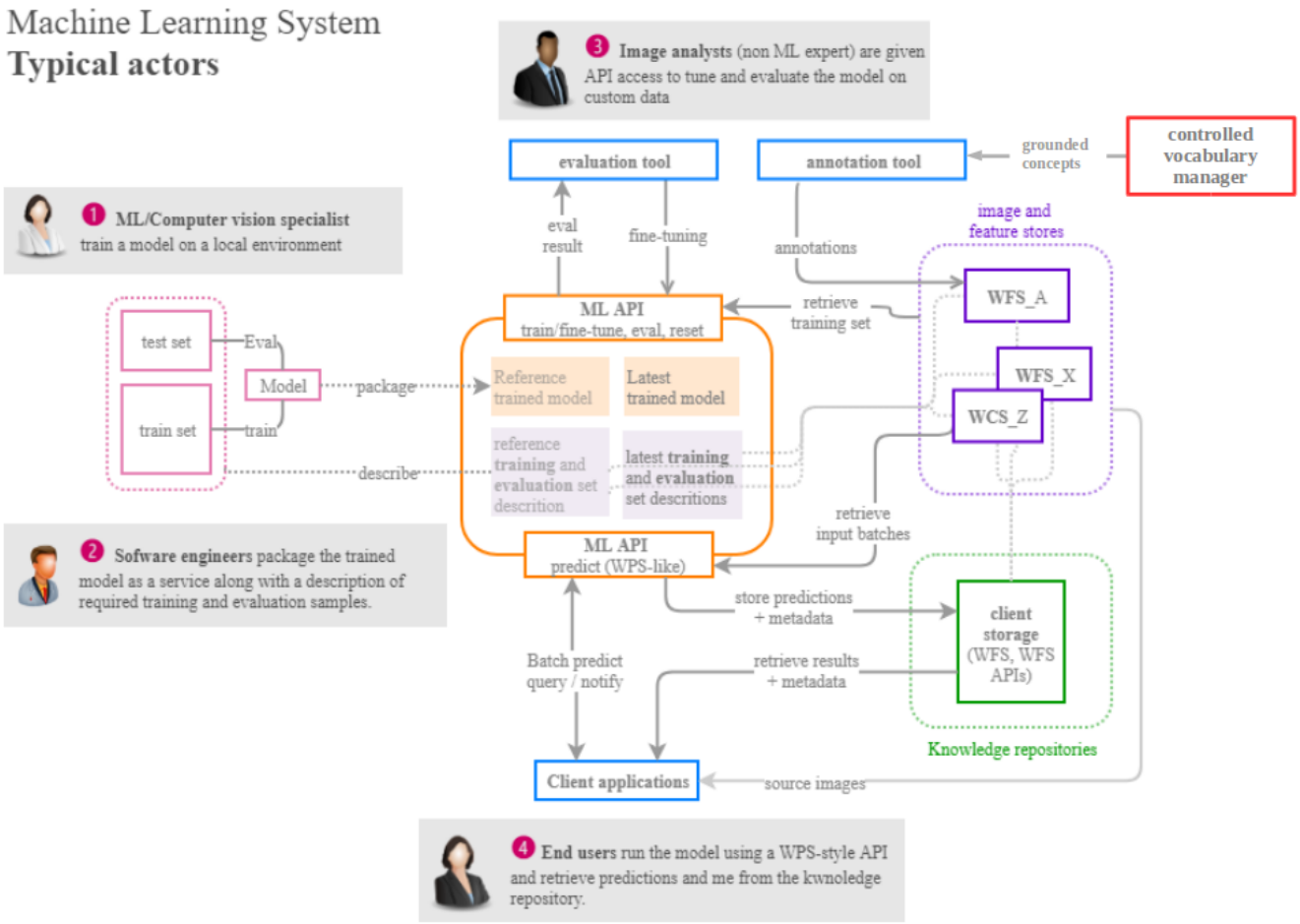

Figure 5. The typical actors in the scenario considered.

Characteristics of the work of an analyst can be found in [26]. Among the major challenges of these users, as noted through operational timelines, are ambiguity in data labelling and trust of models. For the latter, data curation good practices are required but not sufficient for transparency of ML solutions. Additional discussion on assumptions and validations of AI can be found in [11] and [6]. The future analyst is also seen using advanced environments such as Virtual Reality (VR). NGA's 2020 Analysis Technology plan (pdf) [https://www.nga.mil/MediaRoom/PressReleases/Documents/NGA_Analysis_Tech_Plan.pdf] presents some evolution on this important role:

> The role of human analysts will evolve over the next six years as computers assume an active (vice today's mainly passive) role in GEOINT exploitation: human analysts will spend more time interrogating and validating the machine's work, exploiting the gaps, and applying broader contextual awareness to the algorithmic outputs.

6.2. Deliverables

The following deliverables were produced.

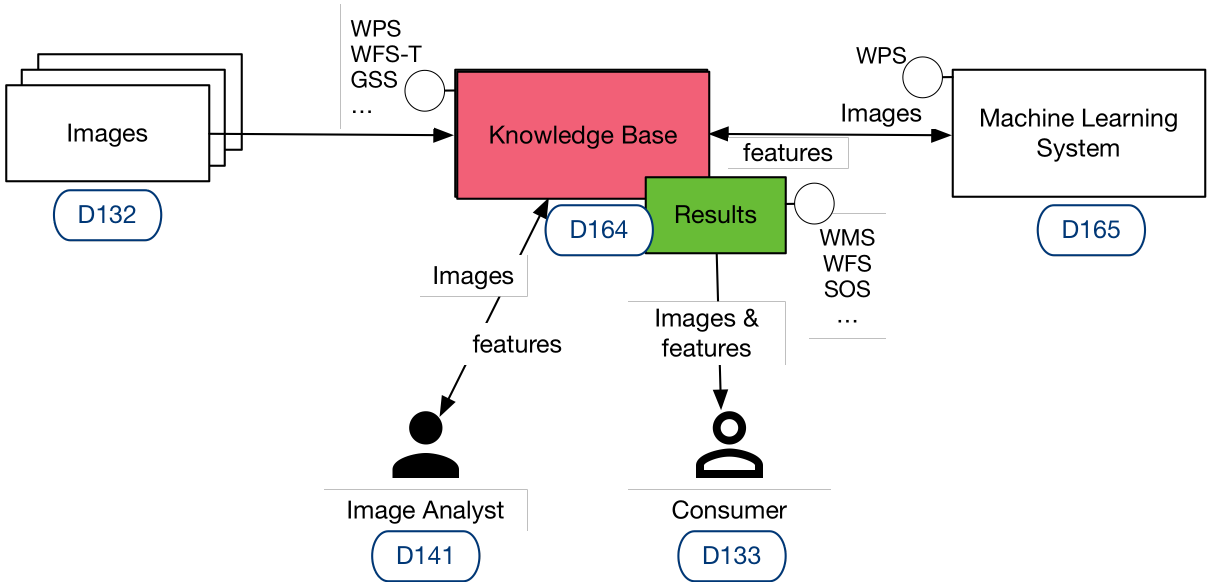

Figure 6. Implementations proposed in the CFP. A new deliverable, D166 Semantic Enablement, was added during the testbed.

6.2.1. D132 Image Repository and Feature Store

This section describes the design and implementation of the actual Image Repository and Feature Store (D132). The major roles of this component are:

- provide source data (imagery) via OGC WFS/WCS
- provide training data (labels) via OGC WFS

The Image Repository and Feature Store is used by other components through these operations:

- *WCS GetCapabilities*
- *WMS GetMap*
- *WCS GetCoverage*
- *WCS DescribeCoverage*

The component, based on GeoServer, stores imagery and feature data and makes it accessible to the Knowledge Base and ML System. The component supports and implements tiling (WMTS). The endpoint is provided on demand for trusted IP ranges.

The WCS holds a PLEIADES scene of Paris. A sample image can be seen in Annex G. Different subsets of this scene were used for training a classification algorithm. The resulting coverages of a classification execution are uploaded to the WCS again via a REST interface.

The image repository offers the following datasets:

- 2 multispectral images (IT,FR) - 2m resolution
- 2 panchromatic images (IT, FR) - 0.5m resolution
- 2 feature files (train and test) containing the annotations (ex: points).

6.2.2. D133 Client to Knowledge Base

This section describes the design and implementation of the client to the Knowledge Base (D133). The major roles of this component are:

- visualization and exploration of the knowledge base and image repository
- explore all imagery, processing results, quality information, and ML validation results

Below is the D133 sequence diagram, which shows the interactions that the client had with the other ML components.

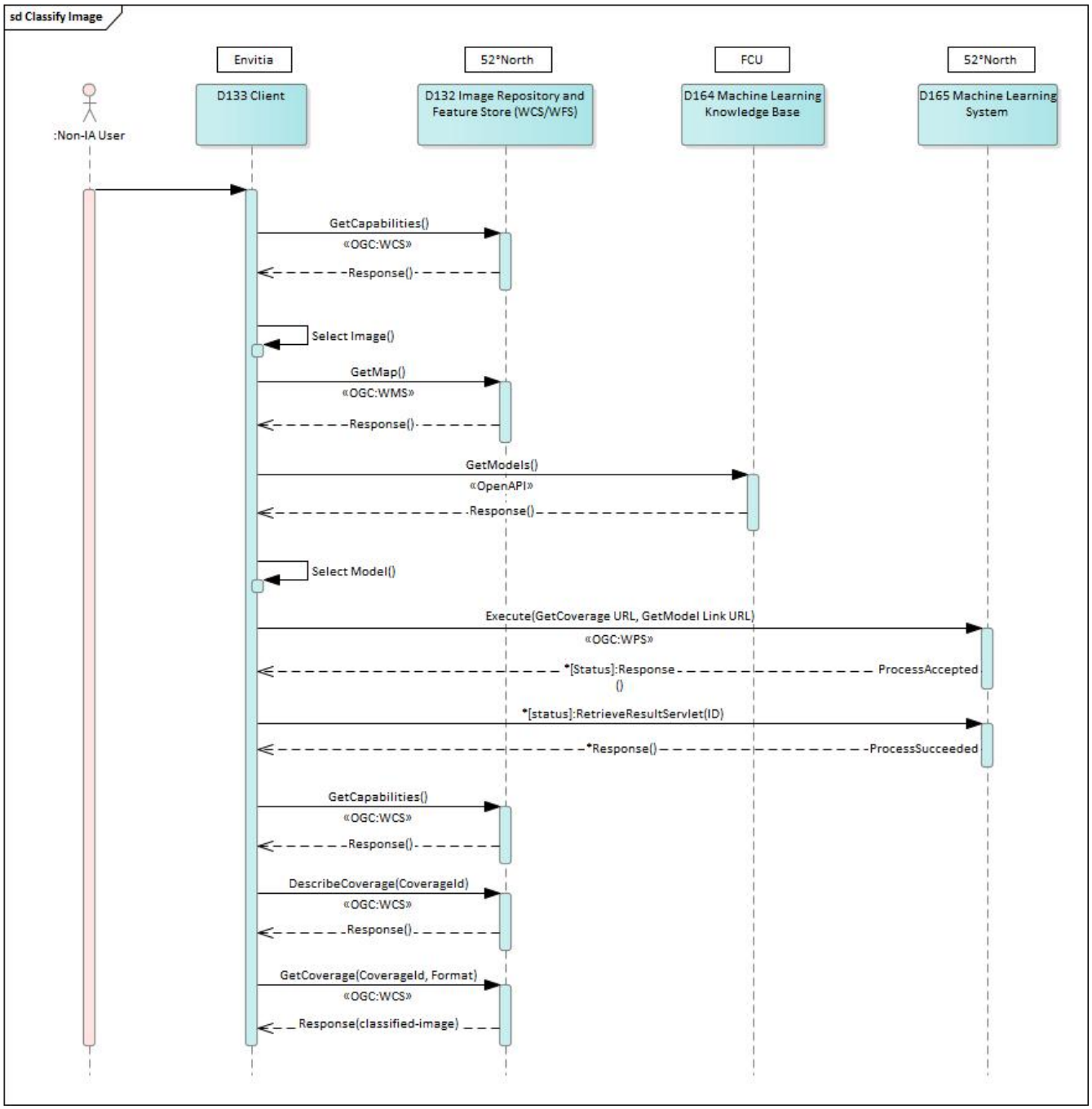

Figure 7. Sequence diagram of the client to KB

Below is the list of steps found in this sequence diagram.

1. Client sends a WCS *GetCapabilities* request to the Image Repository (D132)

2. Client processes the response into a list of coverages/images available
3. User selects a coverage/image to use as the input source data
4. Client sends a WMS *GetMap* request of the selected coverage/image for previewing on the map display
5. Client sends a *GetModels* request to the Knowledge Base API
6. Client processes the response into a list of Models that are available
7. User selects the model to use as the input model
8. Client sends a WPS '*Execute*' request with source-data parameter (using the *GetCoverage* URL) and model parameter (using Link URL)
9. Client receives confirmation that Execute process has been accepted
10. Client iteratively (every 5 seconds) sends a *RetrieveResults* request to the WPS until the response status is '*ProcessCompleted*'
11. Client sends a *GetCapabilities* to the Image Repository (D132) WCS to confirm new image is present.
12. Client sends a *DescribeCoverage* to the Image Repository (D132) WCS so as to get the information needed to correctly request and display the coverage
13. Client uses information from Step 12 to populate a *GetCoverage* request, to request the image back from the Image Repository (D132)
14. Client receives a response with the classified image in PNG format which is displayed on the map display.
15. Not shown in the sequence diagram, but client can also request the model-parameters to be downloaded.

The figure below illustrates the image classification output displayed by the D133 Client to Knowledge Base.

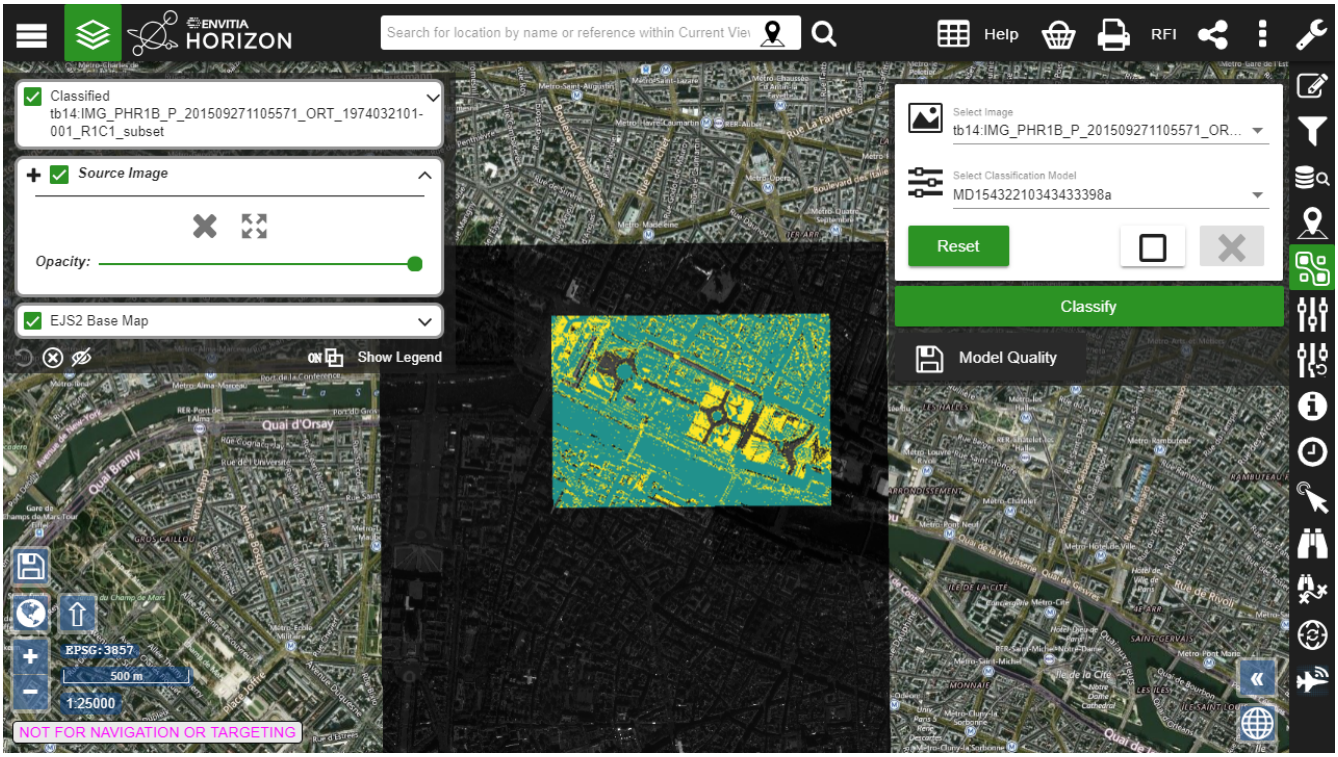

Figure 8. Image Classification Output

6.2.3. D141 Machine Learning Validation Client

This section describes the design and implementation of the ML Learning Validation Client (D141). The major roles of this components are:

- rate the correctness of the ML system output
- make that data available at the knowledge base and as training input to the ML system

Training of models

Below is the D141 sequence diagram, which shows the interactions that the client had with the other ML components when creating a new model through the *Train* process.

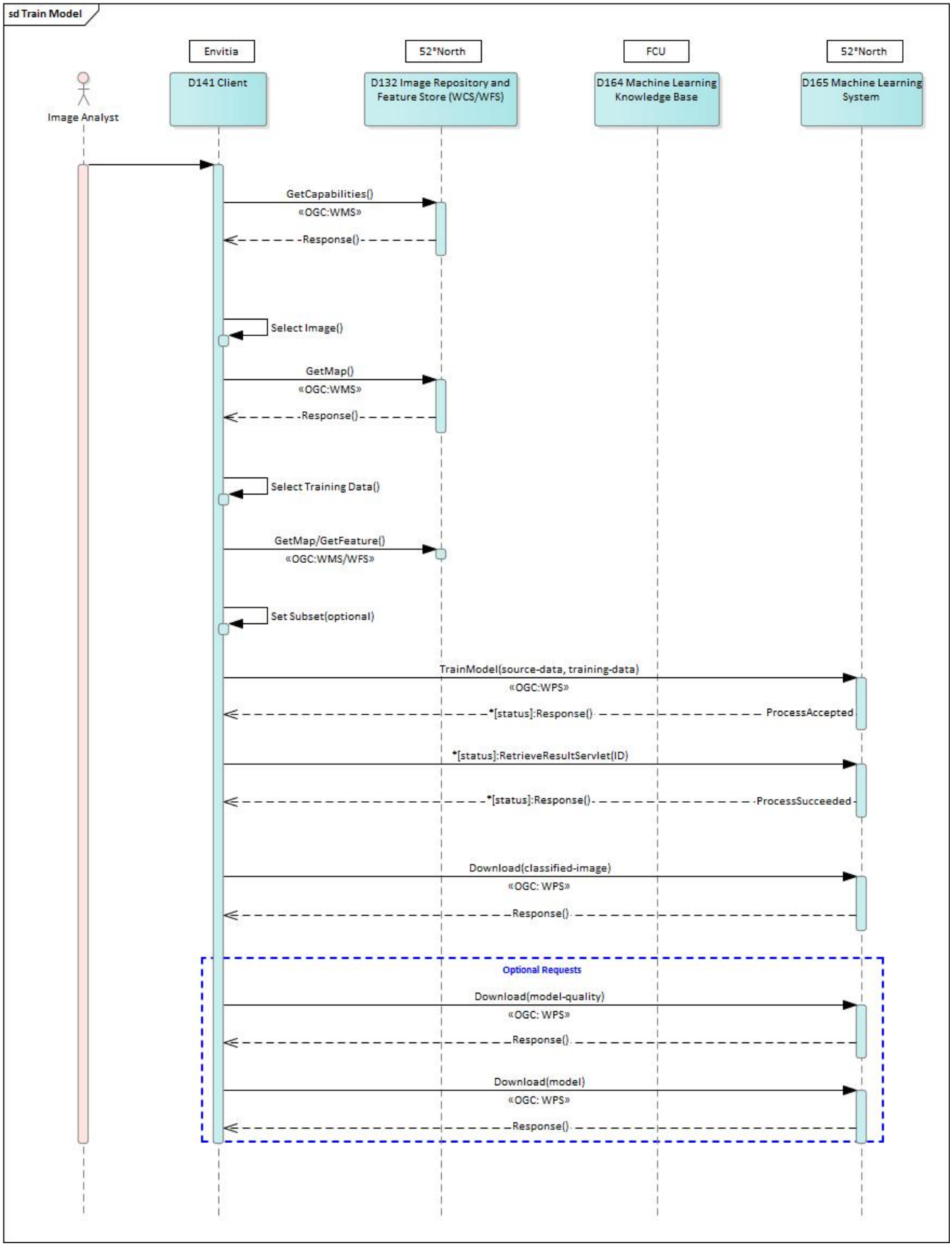

Figure 9. Sequence diagram of the Train operation through the Machine Learning validation client

Below is the list of steps found in this sequence diagram.

1. Client sends a WCS *GetCapabilities* request to the Image Repository (D132)
2. Client processes the response into a list of coverages/images available

3. Image Analyst selects a coverage/image to use as the input source data.
4. Client sends a WCS and WFS *GetCapabilities* request to the Image Repository and Feature store (D132)
5. Client processes the response into a list of training data available to use in the Train process.
6. Client provides ability to crop/subset the source data which is sent to the WPS process
7. Client sends a WPS '*Execute*' request with source-data parameter (using the *GetCoverage* URL) and training-data (using WFS or WCS URL)
8. Client recursively request an update on the status of the WPS and when the response states '*Succeeded*' it retrieves the output.
9. Client receives the resultant classified image from the WPS system
10. The IA optionally selects buttons which instruct the client to download the model quality indicators and/or model of re-trained model.

Re-training of models

The *Train* process implemented by the ML System is used to carry out the re-train process which takes validated features (GML) returned by the IA via WFS-T and use them to re-train the model. Below is the D141 sequence diagram, which shows the interactions that the client has with the other ML components when creating a new model through the *Re-Train* process.

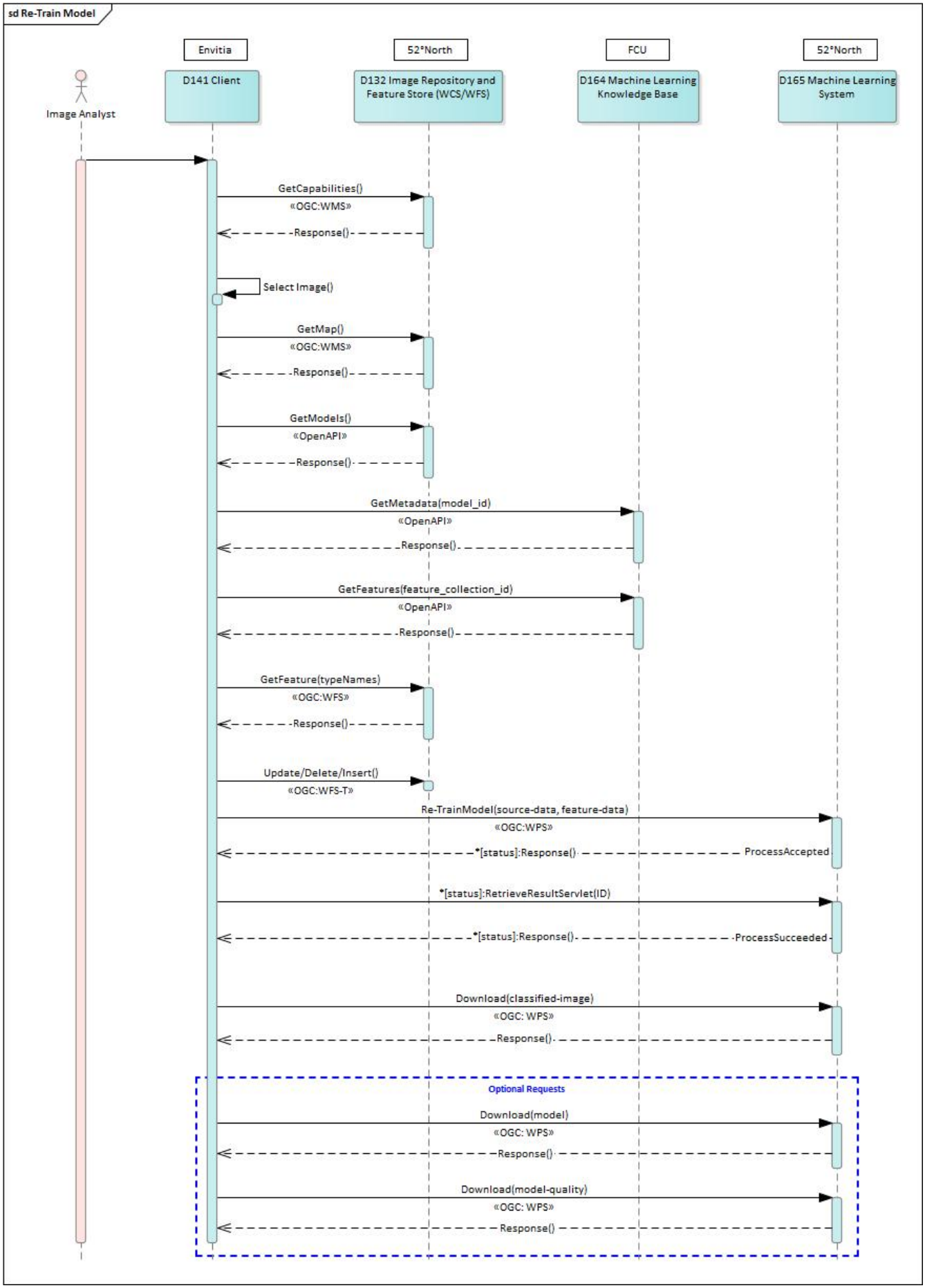

Figure 10. Sequence diagram of the Re-train operation through the Machine Learning validation client

Below is the list of steps found in this sequence diagram.

1. Client sends a WMS *GetCapabilities* request to the Image Repository (D132)
2. Client processes the response into a list of coverages/images that are available User selects a coverage/image to add to map purely for context when validating the classified-features.
3. Client sends a WMS *GetMap* request of the selected coverage/image for previewing on the map display
4. Client sends a *GetModels* request to the Knowledge Base API
5. Client processes the response into a list of Models that are available
6. User selects the model they want to retrain
7. Client makes a call to the *GetMetadata* KB to get all feature collections associated with the selected model (i.e. http://140.134.48.19/ML/GetMetadata.ashx?model_id=<Model_ID>)
8. The client gets the *feature_collection_id* and makes a call to *GetFeatures* (currently this does not work in KB. It is still feature_id). The client now has the URL for the *feature_collection_id*.
9. Client makes an OGC *GetFeatures* request to WFS and adds the layer to the map display in the client
10. Client makes edits to the WFS-T (this could be done in an external GIS, but would be best to have in the client which can link to the vocabulary manager endpoint).
11. There is a lookup tool which makes a call to the Controlled Vocabulary Manager (CVM) so the IA knows which vocabulary to use
12. IA clicks a '*Re-Train*' button.
13. Client uses the same WFS URL and *model_id* from previous steps and executes a WPS Train model process with these URL references.
14. Client recursively request an update on the status of the WPS and when the response states '*Succeeded*' it retrieves the output.
15. Client receives the resultant classified image from the WPS system
16. The IA optionally selects buttons which instruct the client to download the model quality indicators and/or model of re-trained model.

The figure below illustrates the output of the model re-training as displayed by D141 Machine Learning Validation Client.

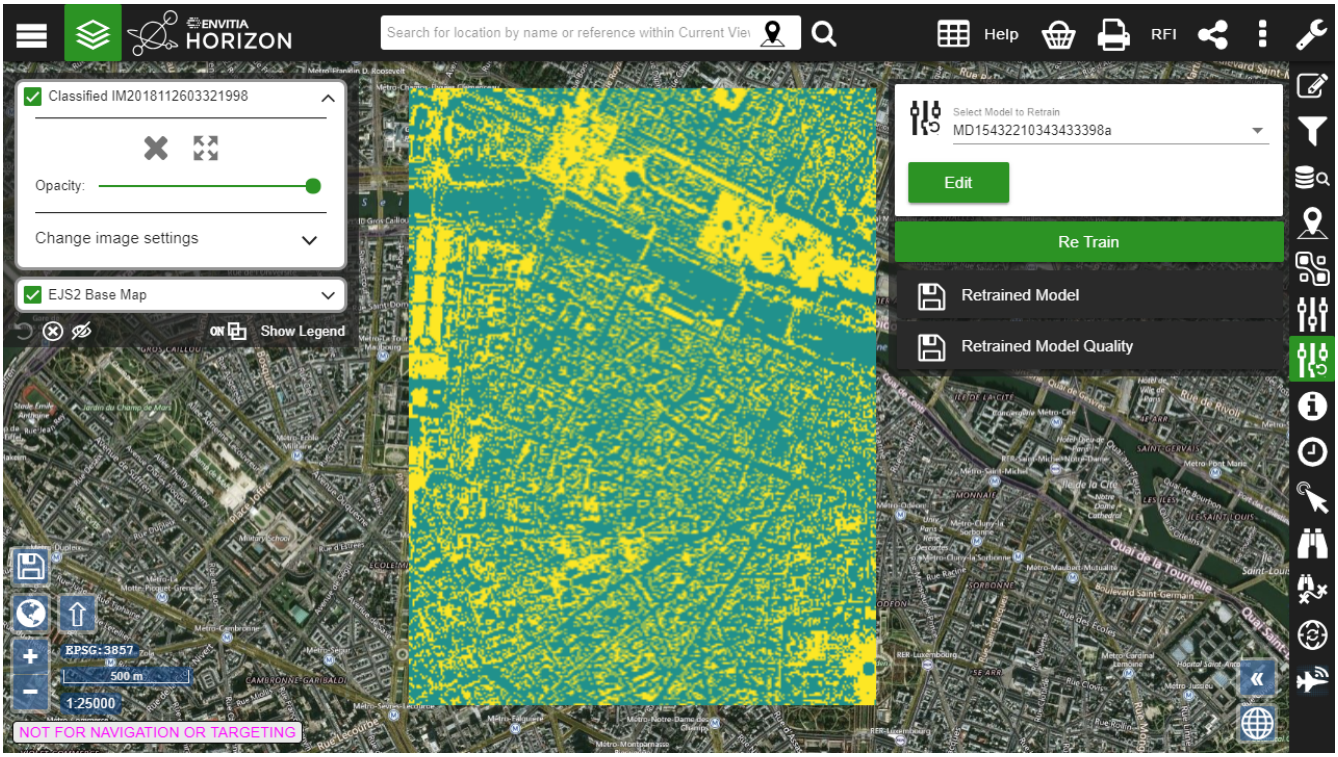

Figure 11. Output of the model re-train process

6.2.4. D164 Machine Learning Knowledge Base

This section describes the design and implementation of the actual ML Knowledge Base (D164). The major roles of this components are:

- store all feature data
- store metadata about models and provenance

The Machine Learning KB provides services to handle the requests between the ML system and the ML knowledge base. The main operations of the KB API are:

- *GetModels* and *StoreModels*
- *GetImages* and *StoreImages*
- *GetFeatures* and *StoreFeatures*
- *GetMetadata* and *StoreMetadata*

Interactions of the ML KB with other components are found in sequence diagrams in the subsections for deliverables D165, D141, and D133.

XML responses

Below are the XML responses for all four Get processes of the KB API.

Subset of XML response for GetFeatures

```
<xml>
  <GetFeatures>
      <Feature>
        <FeatureID>FT2018102204565784</FeatureID>
        <WFS_URL>http://testbed.dev.52north.org:80/geoserver/wfs?SERVICE=WFS</WFS_URL>
        <RequestTime>2018/10/22 16:56:57</RequestTime>
      </Feature>
  </GetFeatures>
</xml>
```

Subset of XML response for GetImages

```
<xml>
  <GetImages>
      <Images>
        <ImageID>IM2018102204565389</ImageID>
        <WCS_URL>http://testbed.dev.52north.org:80/geoserver/wcs?SERVICE=WCS</WCS_URL>
        <RequestTime>2018/10/22 16:56:53</RequestTime>
      </Images>
  </GetImages>
</xml>
```

Subset of XML response for GetMetadata

```
<xml>
  <GetMetadata>
      <Metadata>
        <Metadata_ID>MD2018102204565974</Metadata_ID>
        <model_id>MD154020212074220fd2</model_id>
        <feature_id>FT2018102204565784</feature_id>
        <image_id>IM2018102204565389</image_id>
        <RequestTime>2018/10/22 16:56:59</RequestTime>
      </Metadata>
  </GetMetadata>
</xml>
```

Subset of XML response for GetModels

```
<GetModels>
    <Models>
      <ModelID>MD1540202120742220fd2</ModelID>
      <Title></Title>
      <Abstract></Abstract>
      <Format>Application/zip</Format>
      <Link>http://testbed.dev.52north.org:80/tb14-
d165/RetrieveResultServlet?id=15a3f474-38b0-4f0c-be64-5009f3455cc3model.92552f19-3904-
4d9d-9955-717cde1d4479</Link>
      <RequestTime>2018/10/22 16:56:58</RequestTime>
    </Models>
</GetModels>
```

6.2.5. D165 Machine Learning System

This section describes the design and implementation of the actual ML System (D165). The main operations of the ML system, offered via OGC WPS 1.0, are:

- *TrainML*
- *RetrainML*
- *ExecuteML*

The ML system was delivered with a test client, listed in the components table. When selecting the process in the selection box, inputs are automatically filled and the process is ready to be executed. For the actual execution of the process, the reference to the *model-parameter* output from a training run can be used.

Below are the sequence diagrams for *TrainML* and *ExecuteML*, showing the interactions that the ML system has with the other components.

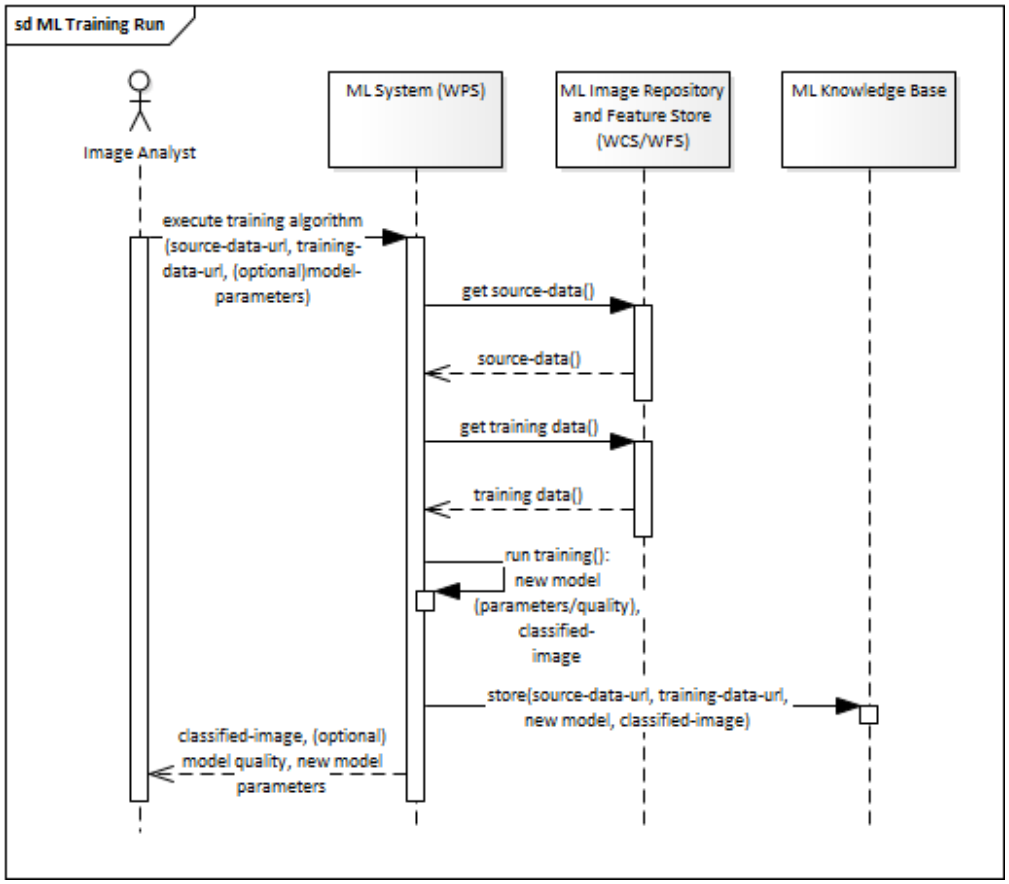

Figure 12. Sequence diagram of the TrainML process

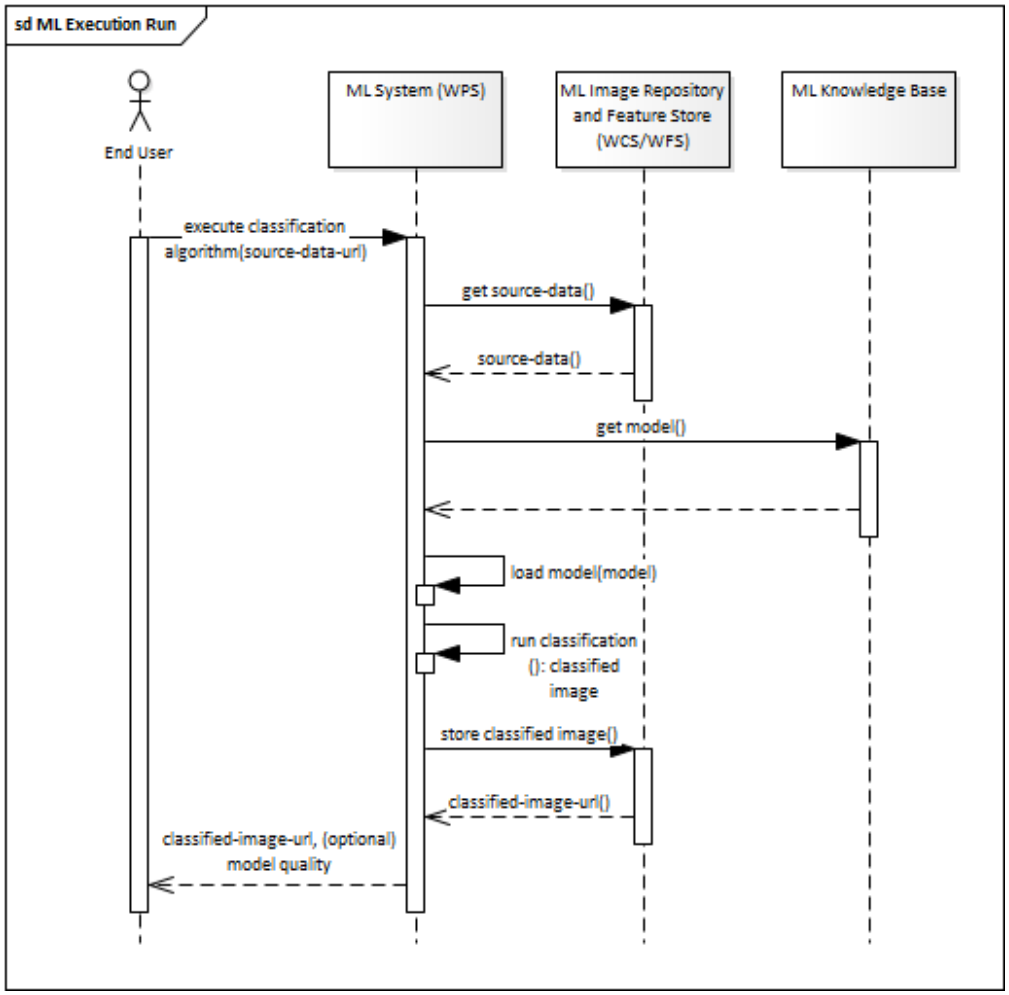

Figure 13. Sequence diagram of the ExecuteML process

System outputs

All three OWS processes provide XML WPS 1.0 and WPS 2.0 interfaces, where a subset showing only one input and one output can be seen below.

Subset of OWS Process Descriptions for D165 ML System

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessOfferings xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/2.0"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
http://schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering processVersion="1.0.0" jobControlOptions="sync-execute async-
execute" outputTransmission="value reference">
    <wps:Process>
      <ows:Title>Execute_MLDecisionTreeClassificationAlgorithm</ows:Title>
      <ows:Abstract>Execute decision tree classification based on Apache
Spark</ows:Abstract>

<ows:Identifier>org.n52.geoprocessing.project.testbed14.ml.Execute_MLDecisionTreeClass
ificationAlgorithm</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Source data</ows:Title>
        <ows:Identifier>source-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Output>
        <ows:Title>Classified image</ows:Title>
        <ows:Identifier>classified-image</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
          <ns:Format mimeType="application/wcs"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

The full process descriptions of *ExecuteML*, *RetrainML* and *TrainML* are found in Annex A, Annex B, and Annex C respectively.

ML system results

The D165 ML system delivered in Testbed-14 uses a decision tree classifier. Below is a description of the algorithm found in Apache Spark MLlib [https://spark.apache.org/docs/2.2.0/mllib-decision-tree.html].

```
Decision trees is a greedy algorithm that performs a recursive binary partitioning of
the feature space. The tree predicts the same label for each bottommost (leaf)
partition. Each partition is chosen greedily by selecting the best split from a set of
possible splits in order to maximize the information gain at a tree node.
```

The algorithm was applied on a Pleiades scene of Paris. The scene was used to gather training data (water areas, pedestrian areas and parks), and to produce a classification image as seen below. Each color belongs to a distinct class that is referenced by the Controlled Vocabulary Manager (CVM) implemented in D166 Semantic enablement of ML.

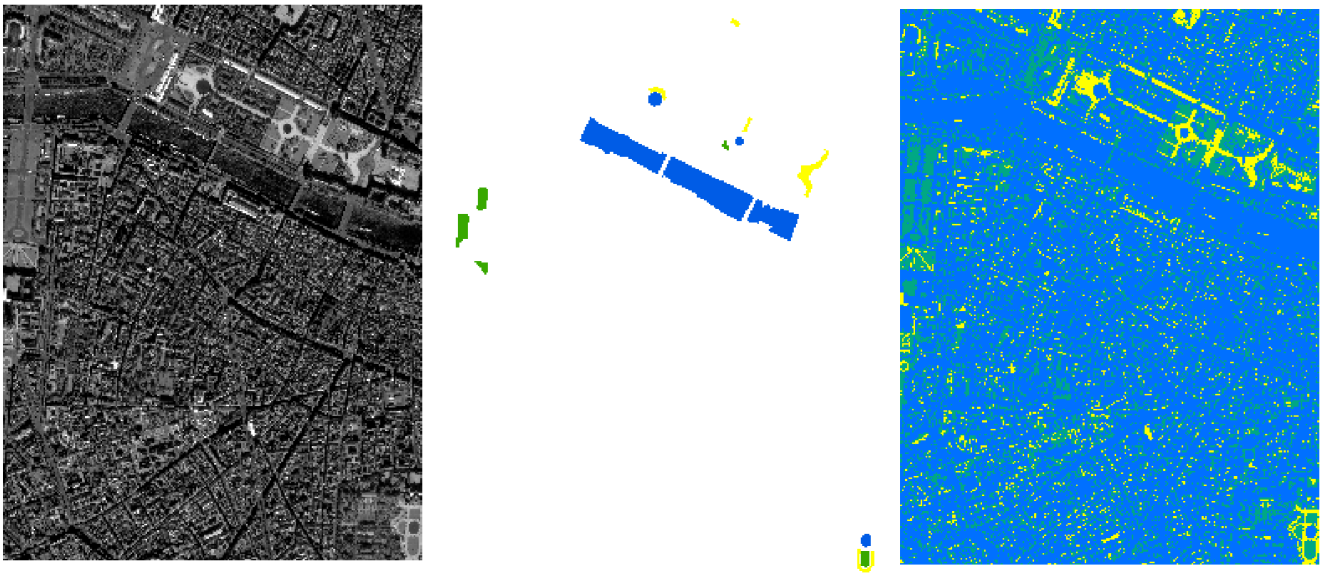

Figure 14. Source imagery, training data and training output from Decision Tree algorithm

For image output results of the actual classification run, please see the Demonstration section. A sample image of the complete Paris image can be seen in Annex G.

6.2.6. D166 Semantic enablement of ML

This section describes the design and implementation of the Semantic Enablement (D166) component. The major roles of this components are to:

- provide an extensible, unified ontology for structured observations offered as a Semantic Enrichment Service
- provide contextually-enriched JSON-LD objects for well-known feature standards (e.g., NEO, NAS) or other reference models (NIEM)

The CVM produces JSON output containing information about vocabulary terms. The CVM offers an endpoint to retrieve terms. The CVM allows searching of terms by:

- *id*
- *uri*
- *namespace*
- *prefix*

- any other field using the *facet* parameter syntax

Controlled Vocabulary Manager use

The caller can request additional, initially hidden fields in search results using the *fields* flag. For example, *fields=conceptScheme* displays the *conceptScheme* field in the search results. To retrieve all fields, use *fields*=*. A full response can be found in Annex E.

Autocomplete text search can be achieved using the *q* parameter. This parameter supports Elasticsearch's query string syntax [https://www.elastic.co/guide/en/elasticsearch/reference/1.7/query-dsl-query-string-query.html#query-string-syntax]. Below is an example URL which searches for the phrase "car":

https://ogctb14.usersmarts.com/terms?q=car

Search within a specific vocabulary namespace can be achieved using the *namespace* parameter. Below is an example URL which searches for the terms within the ConveyanceType namespace:

https://ogctb14.usersmarts.com/terms?namespace=http://api.nsgreg.nga.mil/codelist/ConveyanceType

Search by vocabulary namespace prefix can be achieved using the *prefix* parameter. Below is an example URL which searches for terms with the prefix "codelist":

https://ogctb14.usersmarts.com/terms?prefix=codelist

The CVM also supports search by Views, View Groups, and Bundles. Searching by these fields can be done using the parameters *facet.viewsId*, *facet.groupsId*, and *facet.bundles* respectively. This same pattern, prefixing a field name with *facet.*, can be used to search for Terms by any field with a primitive value. JSON paths are supported.

JSON response

By default, the CVM returns a response like the one found in Annex D. The following is a subset of that response:

Subset of default JSON response of the CVM


```
{
  "results": [
       {
           "id": "4d2fda7b59b1092b2b028f8ddb3d2e01",
           "uri": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/automobile",
           "type": [
               "individual",
               "concept"
           ],
           "primary": false,
           "prefix": "codelist",
           "version": "1.0",
           "label": "automobile",
           "description": "Definition: A motor vehicle generally with four wheels that
carries a small number of passengers.  Description: [None Specified]",
           "namespace": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/",
           "localName": "automobile",
           "qname": "codelist:automobile",
           "category": [
               "Individual",
               "additionalProperties"
           ]
       }
     ]
}
```

Chapter 7. Approaches and good practices

Section 7 presents approaches and good practices as identified by the participants through their own implementations or by referencing authoritative material. The section presents examples of workflows, service interaction patterns, and applications schemas.

7.1. Process profiles for training and execution of models

This subsection presents two profiles considered for the training and execution of ML models.

7.1.1. One Process Profile

In that profile, only one process is specified for training and running the model. The process requires some optional inputs and output parameters, i.e. training returns model parameters/metrics, but the actual execution of a model returns classified output data. Below is a figure showing the sequence of such a profile.

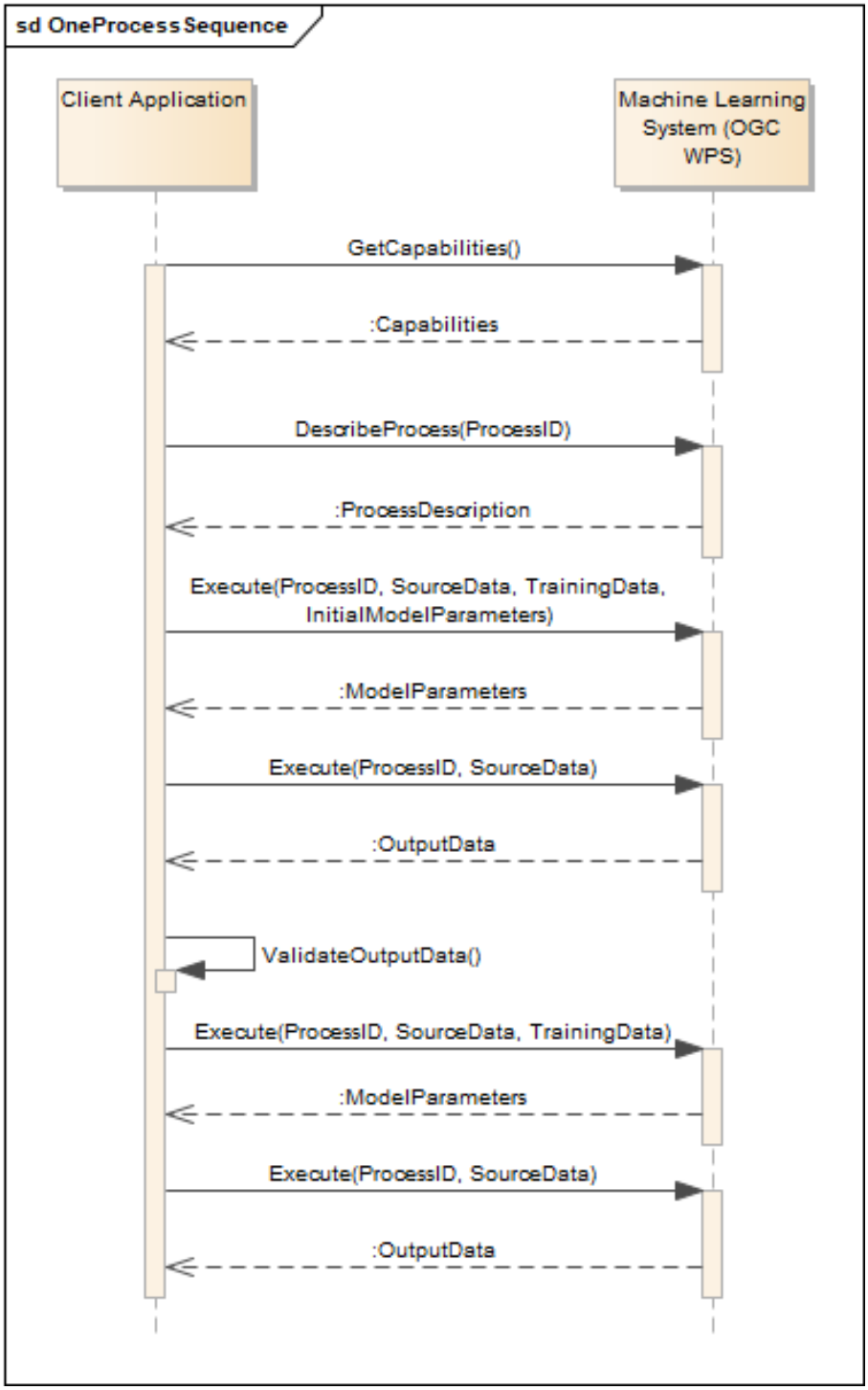

Figure 15. Sequence of training and executing ML models with a WPS offering a single process for both.

- pros: does not break with existing clients
- cons: optional input/output parameter; semantics of the operation specified by setting of some input parameters, e.g. if *trainingData* is passed, then the model is trained.

7.1.2. Two Process Profile

In that profile, one process is specified for training the model and another one for executing the model. The link between the training process and the model execution process needs to be established. The proposed approach is to use a Uniform Resource Name (URN) scheme to describe that link. An alternative approach is to embed the links in the metadata. This profile was deemed the best approach, and therefore was implemented. It was further extended to include another

process for re-training of models. Below is a figure showing the sequence of such a profile.

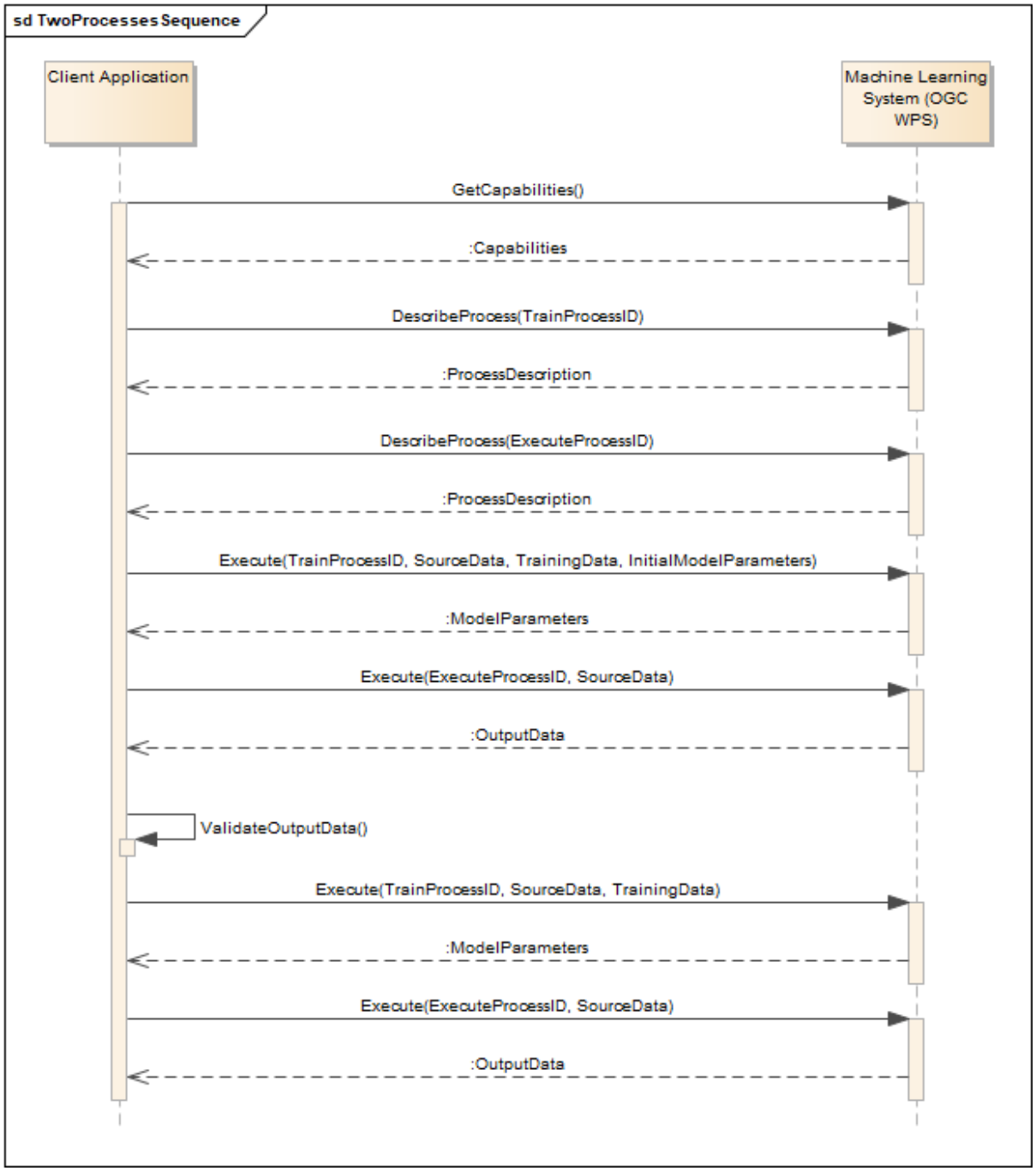

Figure 16. Sequence of training and executing ML models with a WPS offering two Process Profiles for training and executing ML models.

- pros: does not break existing WPS clients; clear semantics of different processes, fewer optional parameters
- cons: link between training and model execution processes needs to be established and managed by client or user

7.2. Additional ML systems

This subsection describes the two other Deep Learning applications designed, implemented, and documented in accordance with the Proof of Concept developed in Testbed-14. The first one relates to VHR imagery semantic segmentation, the second involves transfer of the learning of SAR remote

sensing onto popular DL architectures and trained models.

Table 1. Characteristics of additional models considered in the scenarios

	VHR detector	**SAR segmentation with Spark Deep Learning pipeline**	**Flood detection workflow (EOC thread)**
Task / output	5-classes pixel labelling	Binary ground/water segmentation	Binary ground/water segmentation
Input	High resolution optical imagery patches	SAR patch	SAR images
Method	CNN feature extraction + dense layer for pixel classification	CNN + deconvolutions	Stacking, SFS features + thresholding
Training environment	Pytorch, GPU node	Pytorch, GPU node	-
Inference environment	Command-line tool deployed as a **docker container**	Model used with a batch/streaming query in **Spark cluster**	**Application package**

7.2.1. DL semantic segmentation

This subsection describes the Deep Learning classifier trained on VHR imagery, as found in the components table and demonstrated in the next section.

The image is primarily constructed from the `nvidia/cuda:9.2-base-ubuntu16.04` base image. PyTorch 0.4.0 is added to the construct, as well as a small custom component *dllib* that wraps different functions for training a model or use a model for classification. The DL classifier was delivered as a Docker image that can be run from a single command-line as seen below. It was developed with Application Packaging considerations found in [2].

Command-line execution of the DL detector Docker image

```
#!/usr/bin/env bash
docker run --rm -it \
  --ipc=host \
  --user="$(id -u):$(id -g)" \
  -v /tmp/ogc-pytorch/input:/input \
  -v /tmp/ogc-pytorch/output:/output \
  docker-registry.crim.ca/ogc-public/pytorch-classification:v1 \
  --restore_path /input/MODEL.PTH  \
  --use_gpu 0 --image_path /input/input_512x512.tif \
  --save_directory /output/test_512x512
```

The file MODEL.PTH is in fact a checkpoint from the training/validation process. It contains more than the weights of the network; its content is listed in the following source code block. The *check_point['model_config']* dictionary gives the base architecture to load. Internally, the weights are loaded into the network created from the definition given by *model_config*.

Metadata content of the MODEL.PTH

```
check_point = {}
check_point['state_dict'] = model.state_dict()  # weights
check_point['model_config'] = self.model_config # informations about the architectures
check_point['config'] = config # Training configuration
check_point['epoch'] = current_epoch # Training epoch
check_point['iteration'] = current_iteration # Training iteration
check_point['accuracy'] = current_accuracy # Accuracy
check_point['optimizer_state_dict'] = self.optimizer.state_dict()  # optimizer params
check_point['criterion_config'] = self.criterion_config # cost function
check_point['optimizer_config'] = self.optimizer_config  # optimizer def
```

The labels for each class are determined from the *config* section of MODEL.PTH. Five classes (cars, trees, roads, houses, industrial buildings) were used. About 1000 points per class was used for the training set, on a single Pleiades 50 cm image covering the Vancouver area. Instead of using a variable patch size, for instance a bounding box drawn by a user, this particular implementation uses a parameterized single patch size to extract training features. The classification model operates on a per pixel basis. The process returns multiple outputs: the classification image (8uint) and the probability image for each class (32f).

7.2.2. DL transfer learning

This subsection describes the image preprocessing and flood detection application packages delivered in the Earth Observation & Cloud (EOC) thread of Testbed-14. This practice is relevant to the ML engineering report because it documents a valid pre-processing workflow for ML systems composed of a stacker and a feature generator. This practice also presents a valid Earth Science process, a flood detector, that could benefit from packaging and linking of ML systems as well as access to interoperable geospatial ML-targeted Open APIs.

This application is twofold:

- A flood detection application package based on a **domain expert workflow**. This package encompasses SAR image pre-processing, feature extraction (SFS descriptor), and the final segmentation.
- An **experimental Deep Learning model** that can be used within distributed Apache Spark Queries. This model was trained to reproduce results from the domain expert workflow.

The diagram in Figure 3 summarizes the relationship between the two models.

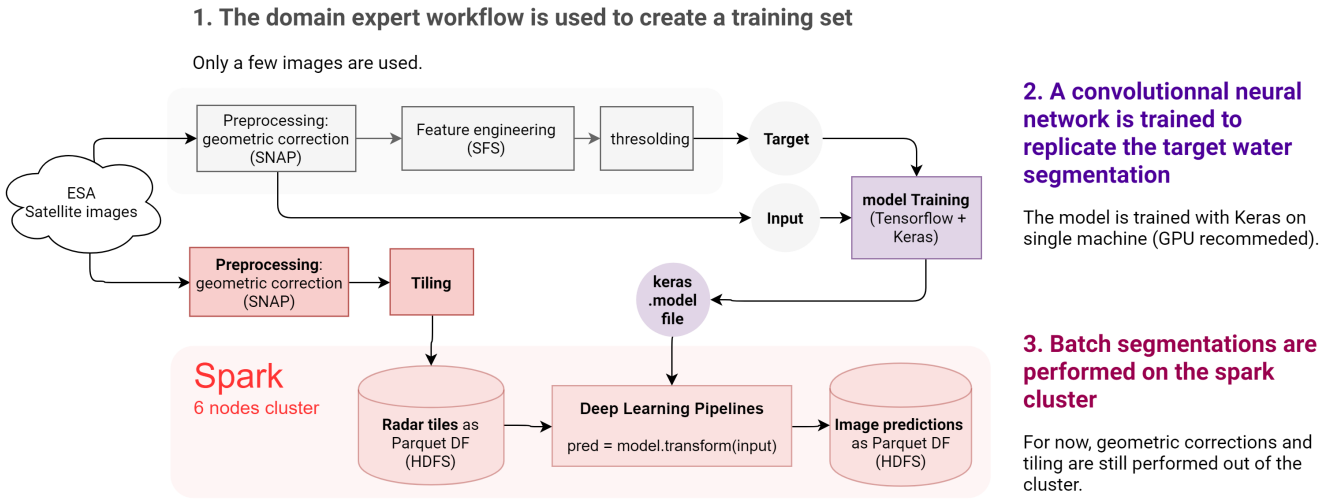

Figure 17. Domain expert workflow for SAR image segmentation (gray) and experimental neural network model deployed in Spark (red).

Apache Spark has risen as one of the most important tools of big data analytics in a distributed computing environment. While providing a simple programming model, based on map-reduce operations, Spark follows the general big-data paradigm "bring the processing to the data" (data locality principle). This is a particularly relevant for earth observation and telemetry tasks where the processing is relatively uniform and applied to large images.

The DL model is trained on a local machine, using the PyTorch framework. It leverages **transfer learning** by reusing parameters from a pre-trained neural network trained on the ImageNet dataset (see Figure 18).

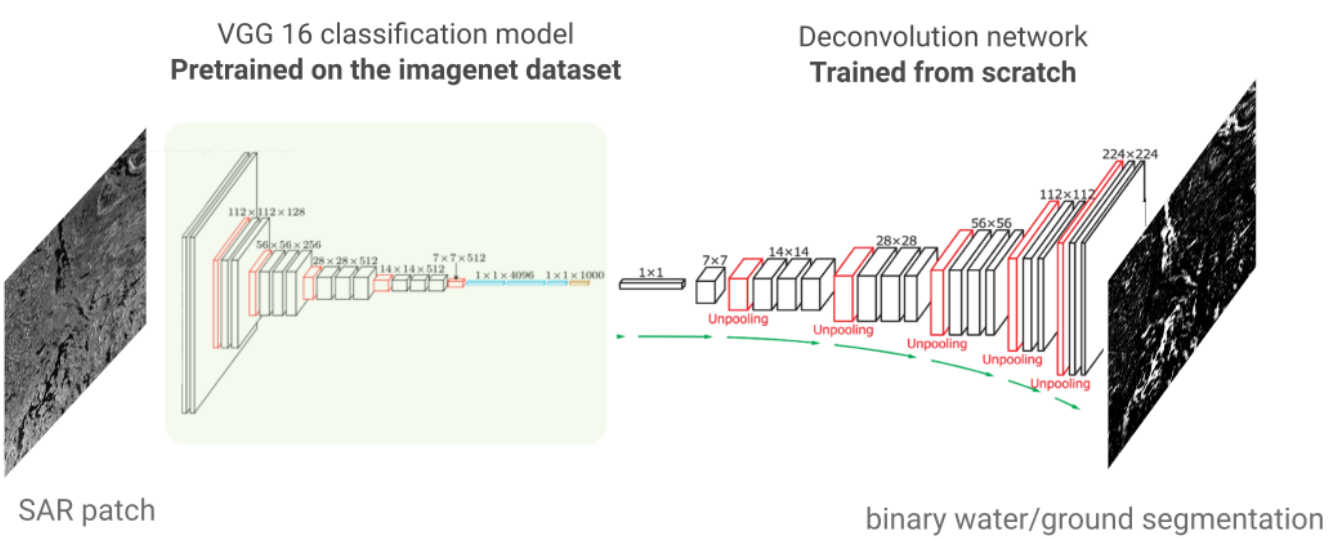

Figure 18. Deep learning model architecture : pretrained VGG16 and deconvolution.

7.3. Application schemas

This section presents the metadata schemas proposed by the participants and used in their Technology Integration Experiments (TIEs).

GML application schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ogc-tb14-ml=
"http://schemas.opengis.net/testbed14/machine-learning" xmlns:gml=
"http://www.opengis.net/gml/3.2" targetNamespace=
"http://schemas.opengis.net/testbed14/machine-learning" elementFormDefault="qualified
">
    <xs:import namespace="http://www.opengis.net/gml/3.2" schemaLocation=
"http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>

    <xs:element name="ClassifiedFeature" type="ogc-tb14-ml:ClassifiedFeatureType"
substitutionGroup="gml:AbstractFeature">
        <xs:annotation>
            <xs:documentation>A feature representing a real-world object that has been
classified by a Machine Learning image classification algorithm.</xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:complexType name="ClassifiedFeatureType">
        <xs:complexContent>
            <xs:extension base="gml:AbstractFeatureType">
                <xs:sequence>
                    <!-- The feature identifier in the Feature Store. -->
                    <xs:element name="Id" type="xs:long"/>
                    <!-- The image classfication category, ideally from a controlled
vocabulary such as the NAS or NIEM. -->
                    <!-- Implemented here as xs:string for simplicity. Could use
dedicated GML property for codelists instead. -->
                    <!-- Similar to properties tb14:category and tb14:term on
tb14:labels feature type in current Feature Store WFS. -->
                    <xs:element name="classificationType" type="xs:string"/>
                    <!-- The geometric footprint of the feature. -->
                    <xs:element name="the_geom" type="gml:GeometryPropertyType"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

enhanced schema of ML processes to use and produce features

```
<xsd:schema xmlns:gml="http://www.opengis.net/gml" xmlns:tb14=
"http://www.opengeospatial.org/projects/initiatives/testbed14" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace=
"http://www.opengeospatial.org/projects/initiatives/testbed14">
<xsd:import namespace="http://www.opengis.net/gml" schemaLocation=
"http://testbed.dev.52north.org:80/geoserver/schemas/gml/2.1.2/feature.xsd"/>
<xsd:complexType name="ml-trainingdata2Type">
  <xsd:complexContent>
<xsd:extension base="gml:AbstractFeatureType">
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="0" name="the_geom" nillable="true" type=
"gml:MultiPolygonPropertyType"/>
<xsd:element maxOccurs="1" minOccurs="0" name="term" nillable="true" type="xsd:string
"/>
<xsd:element maxOccurs="1" minOccurs="0" name="value" nillable="true" type="xsd:int"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="ml-trainingdata2" substitutionGroup="gml:_Feature" type="tb14:ml-
trainingdata2Type"/>
</xsd:schema>
```

The CFP also lists the following schemas and references:

- Multinational Geospatial Coproduction Program (MGCP)
- NSG Application Schema (NAS)
- NSG Core Vocabulary

Chapter 8. Demonstration

Section 8 demonstrates the proof of concept (POC). The demonstration scenarios and relevant material are presented.

8.1. Results

The demonstration of the current work can be considered a success. The client application was indeed shown to execute processes offered by the ML system and display its results found in the Image and Feature Repository. As expected, the ML system provided classification results that in turn were referred to in the KB. The proposed schemas did integrate controlled vocabulary references.

Participants diligently interacted through the GitHub issue tracker [https://github.com/opengeospatial/D030-Machine_Learning_Engineering_Report/issues?utf8=%E2%9C%93&q=is%3Aissue] in order to describe and solve problems and complete their interoperability experiments.

Difficulties in the followings aspects were noted, all mostly due to technical complexity:

- creation of data models capturing the whole annotation, training, and retraining process
- production of a controlled vocabulary that supports demonstration scenarios, both for land covers and individual objects
- integration of ML model outputs into referenceable WFS layers and features
- rapid creation of accessible endpoints to be tested by other participants
- integration of a DL detector for VHR imagery in the form of an Application Package

It is expected that through future deliverables proposed in the future work subsection of the summary, these difficulties would be solved in early stages of a subsequent testbed.

The reader can find additional hints on problematic, outstanding or limited elements in discussion and open issues.

8.1.1. Horizon WebClient

Participants agreed that demonstration of the ML task work is better served by the client. Here, the engineering report describes the major functionalities developed through the User Interface (UI) of the client.

The Envitia Horizon Portal is an advanced, OGC conformant web-based user interface which can be used with most modern browsers to provide exploitation of geo-enabled services. Envitia Horizon provides the ability to find, visualize, and retrieve map datasets, as well as use of a suite of analysis tools.

A core concept within Envitia Horizon is the idea of a community of interest (COI). A COI is a group of users that may share similar interests. From an administration perspective, a COI can be specifically tailored to a community's needs by enabling the analysis and data visualization tools on a per-COI basis. The portal therefore uses this concept to manage which tools the user has access to

(the simple end user and the Image Analyst).

Classify Image Model

The screenshot below illustrates the functionality of the tool which runs the image classification model.

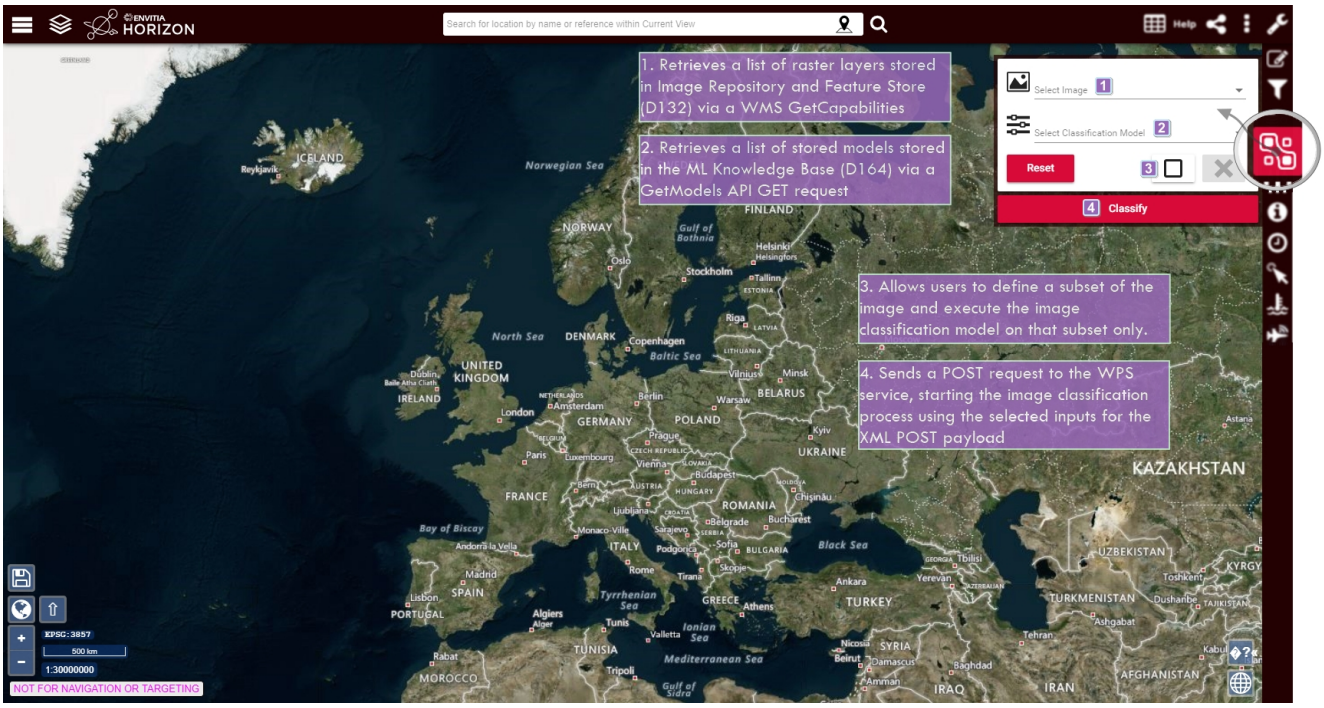

Figure 19. Envitia Image Analyst Client – Defining the Image Classification Request

As soon as the user selects a source image (1), the client sends a WMS *GetMap* request to the Image Repository/Feature Store service. This is so that the user can preview and confirm the image on the map display. An example of this request is as follows:

http://testbed.dev.52north.org:80/geoserver/ows?SERVICE=WMS&VERSION=1.3.0&REQUEST=GetMap&FORMAT=image%2Fpng&TRANSPARENT=true&LAYERS=N52%3Aprimary2166771569326150765.tif_24912ec6-4464-4ccf-82a8-d3f0cb9fd2f6&STYLES=&BBOX=1472482.912885636%2C5919283.470404049%2C1477374.8826958875%2C5924175.4402143005&WIDTH=256&HEIGHT=256&CRS=EPSG%3A3857

The screenshot below illustrates the functionality of the tool which runs the retraining of a model.

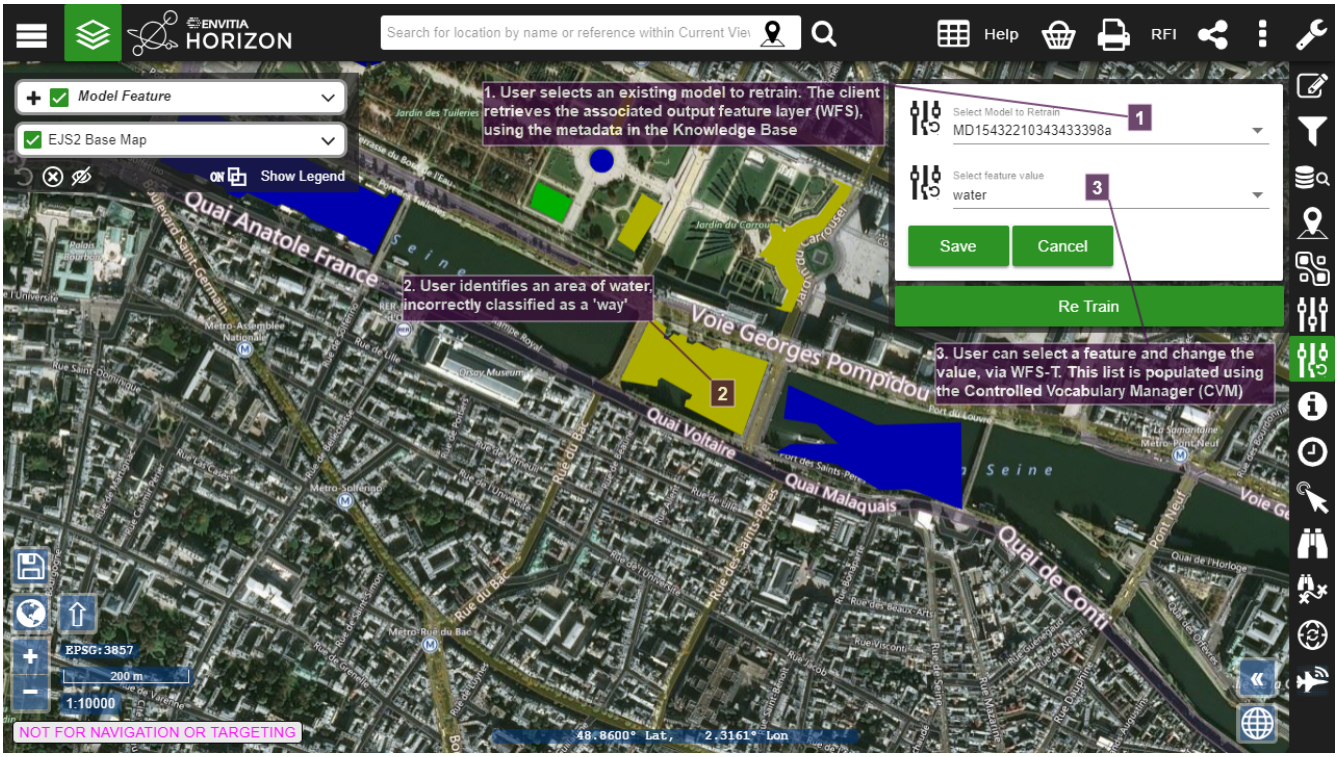

Figure 20. Envitia Image Analyst Client – Re-Training the Model

8.1.2. ML classification

Different subsets of the PLEIADES scene were used as seen below.

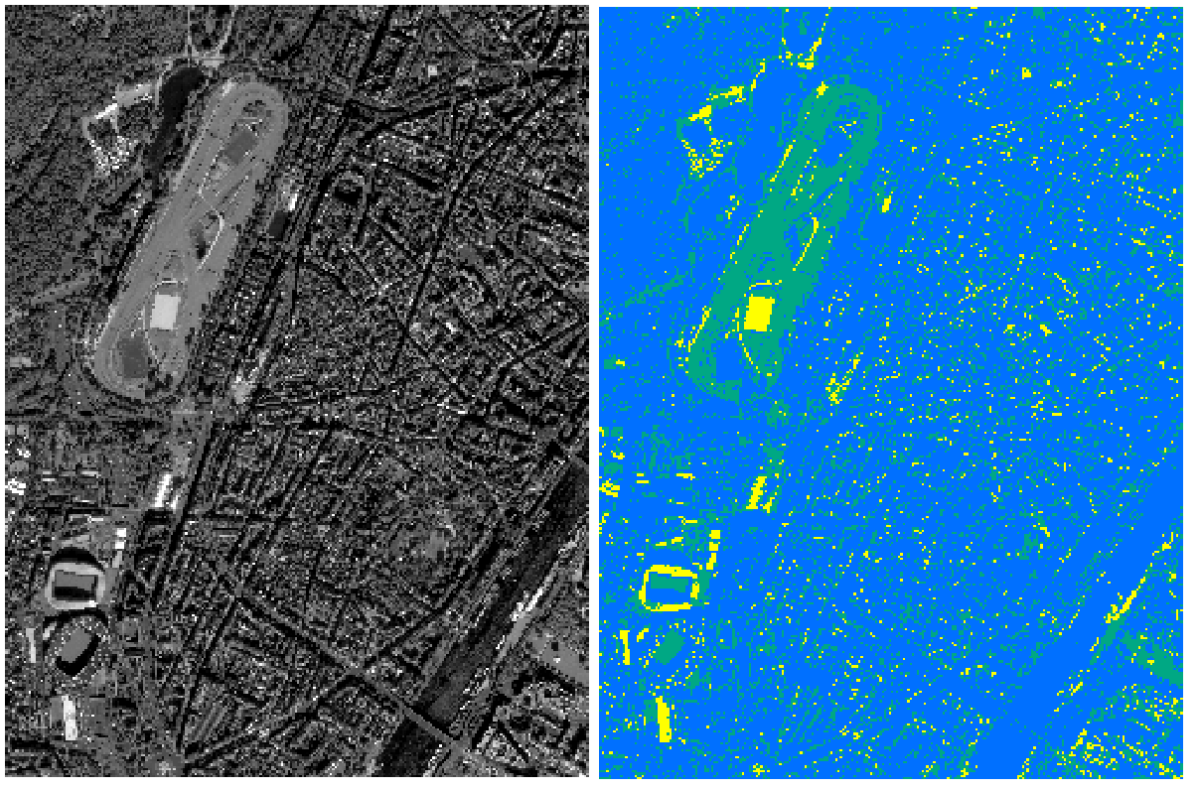

Figure 21. Source imagery and classification output from Decision Tree algorithm

8.1.3. DL semantic segmentation

Below is the output from the DL classifier trained on Pleiades 50 cm VHR imagery. Five classes

(cars, trees, roads, houses, industrial buildings) were used.

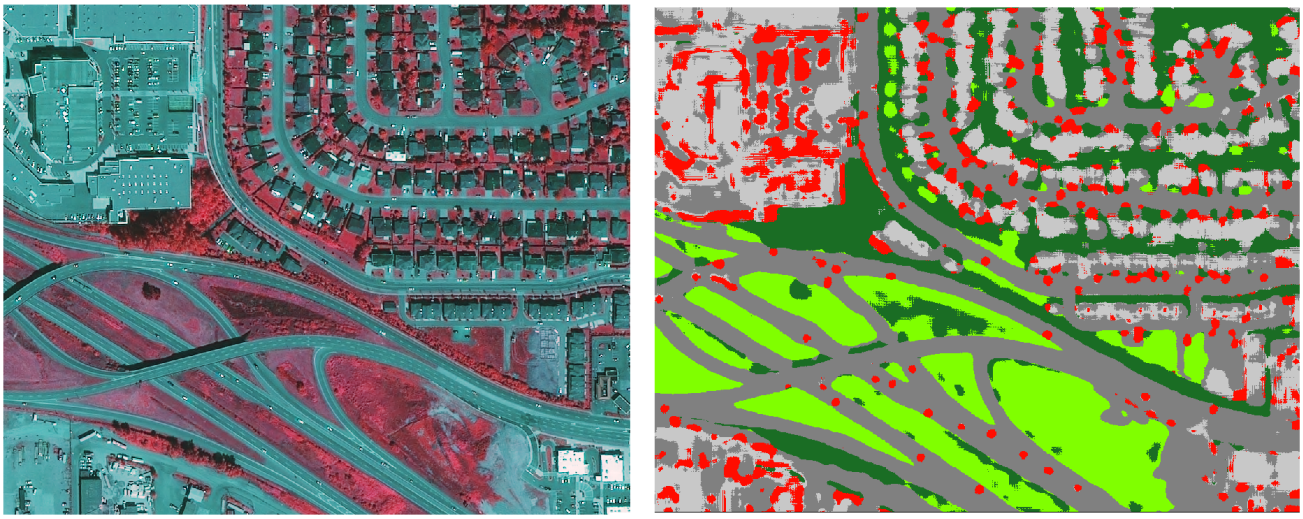

Figure 22. Pleiade VHR imagery of the city of Vancouver and its associated semantic segmentation, produced by a Deep Learning detector delivered in the testbed. Image courtesy of Effigis Geo Solutions. Includes material © CNES 2018 (year of production), Distribution Airbus DS Geo SA / Airbus DS Geo Inc., all rights reserved

Chapter 9. Discussion and Open Issues

Section 9 opens up the discussion by further elaborating on future work recommendations and raising open issues from the ML task. In general, these recommendations address the creation or extension of OGC standards to support ML and AI geospatial algorithms.

9.1. Knowledge base as a NextGen catalog

The Machine Learning Knowledge Base is effectively a catalogue of metadata about models, available images, available features, and information on previous model execution/training. In an OGC context, CSW could be used as a standardized interface in the Knowledge Base. This should also consider whether the information stored in the Knowledge Base could be handled using the OGC CSW-ISO Application Profile of CSW, which utilizes the ISO 19115 metadata model. If the ISO 19115 model is not rich enough to support the Knowledge Base use cases, then a more extensible registry model, such as the one offered by the CSW-ebRIM Application Profile, could be considered.

The Testbed-14 Next Generation Services (NextGen) thread addresses both new capabilities at the service level as well as complex interaction patterns between services. One of the tasks is to investigate Next Generation OGC Web Services RESTful interfaces by experimenting with the new WFS 3.0 specification and to add security mechanisms based on OAuth2.0. The use of the OpenAPI style of interaction in the ML Knowledge Base API is similar to the approach being investigated in NextGen. In order to forge links between the ML task and NextGen tasks, it is recommended to experiment with Next Gen CSW services using the findings from Testbed 14 on WFS 3.0.

9.2. Application packaging & WPS 2.0

In line with Testbed-14 EOC Application Packaging best practices [2], it is recommended that future work include packaging of ML systems for major AI framework. One advantage is the easier discoverability, distribution and management of various ML systems such as TEP and MEP architectures [32]. The major systems described in this work, the Execution Management System (EMS) and the Application Deployment and Execution System (ADES), both implement WPS-T 2.0 REST/JSON bindings. Additionally, work conducted in the EOC thread of Testbed-14 describe best practices for workflows, including data fetching, data preparation, creation of handcrafted features, publication of outputs, etc.

9.3. Segmentation and classification of ML outputs

A post-processing step is required to segment the resulting regions into individual classified features (e.g., a car or a park) that can be stored with WFS. For the ML system, this can be achieved by transformation of the output into *blobs*. For the DL detector, the point location of an object is determined by the mode (maximal value) of each blob of the probability map for each class, but the contours remain fuzzy.

Several approaches to classification assume a flat structure to labels. Performance of DL can be significantly increased by combining popular datasets. This can be achieved in several ways. For example, using a hierarchical tree generated from WordNet, labels from ImageNet and COCO can be merged. It is recommended to continue experiments aiming to merge annotation datasets.

9.4. Metadata

Supervised learning algorithms implemented and described in the testbed for ML system include Decision Trees and Deep Neural Networks. Once trained, both the outputs of both methods can be described as models, but their parameters, associated metadata, and performance differ significantly. In most cases, training time for a specific dataset and inference time for a given input are applicable. For performance, precision and recall for a given input are also applicable when ground truth data is available. It is recommended that participants consider adding training time, inference time, precision, and recall in the interfaces of their deliverable to better demonstrate use in a disaster scenario or, more generally, in GeoInt use cases.

9.5. Temporal enablement

In the Testbed-14 ML task, the participants developed a descriptive analytics use case in the spatial domain: "What has happened here? What do we see?". The scenario did not make extensive use of data with variety on the temporal axis. It is recommended that future experiments in Machine Learning includes a support for a variety of temporal data, potentially including IoT timeseries, satellite imagery, weather forecasts, climate projections, etc. Such variety enables predictive and ultimately prescriptive scenarios that are highly valued to AI users. Support for a wide array of temporal data will also advance geospatial standards, for example by establishing good practices for timestamp management of data or catalog querying and filtering. For example, Testbed-14 also looked at swaths - vertical collections of atmospheric wind velocity. Each "pixel" is a stack of wind direction and magnitude measurements at a specific place and time.

9.6. Model interoperability

Interfaces developed in the testbed for ML systems favor interoperability at the data level (get source data, get training data, store) and at the process level (train, classify). Currently, no interoperability is possible at the model level (load model, get model), as algorithms such as Decision Trees and DNNs, both applicable as classifiers, are fundamentally different in their structure. Initiatives like the Open Neural Network eXchange (ONNX) aim to enable interoperability of models into various popular AI toolsets and frameworks [1: https://onnx.ai/]. It is recommended to consider publishing neural network capabilities of ML systems, thus allowing richer use cases, for example exportation from one ML system to another one through a workflow via *get model* and *load model* operators.

Implementations in the ML task used an Image Repository mainly to store data inputs and inference outputs of the model, facilitating their retrieval. In order to increase transparency and ultimately improve trust of algorithms, it is suggested to experiment with the use of OWS to facilitate visualization and interaction with the internal layers of CNNs. This assumes that artificial neural network models' weights and architecture can be assimilated to structured, dense spatial data (WMS, WCS) containing patterns and features (WFS) used for AI's decisions or recommendations for a specific query (WPS).

- Develop appropriate styling to enable pattern discovery, for instance color palettes (WMS) or styling of features (WFS);
- Develop a process (WPS) that allows exportation of deep features directly into image

repositories (WMS) or datacubes;

- Develop a process (WPS) that allows crude visual inspection (WCS) of a model's internal parameters;
- Develop a process (WPS) that allows creation by an analyst of features in a model (WFS) to highlight deficiencies or efficiencies for specific input, thus facilitating rejection or adoption of the model in certain situations.

9.7. Semantic interoperability

The Controlled Vocabulary Manager (CVM) provides a service for finding terms by various parameters. The service is used by the Machine Learning Clients to provide a list of terms for classifying features in an image. The CVM could be fully integrated into the OGC AI/ML architecture and used to provide a common vocabulary to all components within the architecture: ML System (WPS), Knowledge Base, Machine Learning / Image Analyst Clients, and Image/Feature Store. Other topics for consideration include standardization, from an OGC context, of the service interface and data exchange formats used by the CVM service, for example potential use of the OGC Catalogue Service interface (CSW) and data encodings such as GML.

Inline with Testbed-14 EOC Execution Management System (EMS) best practices, consider exploration of advanced platform-based use cases for AI. GeoInt literature describes "query interpreters" and subsequent "query conductors" for federated subsystems. Query interpreters use NLP, a form of AI, to understand text-based user queries and automatically produce federated workflows. Currently, work in EOC describes use of graphical workflow editors. Future work for EMS could include experiments on automatic generation of workflows or selection of catalogued workflows that are closest to a user query, thus advancing "query interpreter" concept. Experiments in the creation of *Analysis Ready Data* and *Dynamic Analysis Ready Data* could support such efforts.

9.8. Tiles

Experiment with use of tiles (WMTS) to directly feed training patches for DL. Several DL implementations for supervised semantic segmentation use raw GeoTiff images as input. Patches are often extracted from generating a bounding box around a labelled pixel. An experiment using tiles would either allow annotations at the tile level or allow efficient fetching of tiles belonging to a labelled feature (point, region). Considering that efficient subsetting and fetching of imagery data is one of the biggest advantages of using tiles, this suggests an experiment involving the use of OGC Web Services such as WMTS.

Tiles can also enable efficient DL training for several levels of resolution. In this experiment, annotations made on VHR imagery could be cascaded to lower resolutions. In such an experiment, there are tradeoffs to the use of tiles. For instance, only 3 bands are fetched per request by default, limiting usefulness with multispectral imagery. Bias can be introduced by data preprocessing, hindering use for scientific studies. Consider extensions to WMTS to support larger number of bands or to describe metadata related to image pre-processing.

9.9. Virtual environments

Experiment with the use of 3D tiles in Augmented Reality (AR) or Mixed Reality environments in order to produce interoperable annotations between 2D, grid-based annotations and their 3D environment counterparts. For example, assuming grid-based annotation are more mature, an experiment using 3D tiles could allow filtering, retrieval and visualization of 2D annotations produced in spatial proximity of the AR user. The user could then enrich the labels according to his tasks, either by adding an elevation component or triggering image capture from the ground. Also, assuming labelled ground-level imagery is abundant (Flicker, Google Image, etc.) and related ML practices in the literature abounds, the image capture from AR users could be structured similarly and be consumed by classical computer vision models.

An experiment in an AR environment could also benefit from semantic contextualization. For instance, if the user is a humanitarian worker on the field during a disaster scenario, he/she would be interested in recent satellite imagery annotations in his area related to structural damages, injuries or logistics.

9.10. Data management

Every time a user runs an image classification model, the output image is automatically sent into the Image Repository – meaning it comes up as a source/training target for the next user. This is not a significant issue in a demonstration environment, but a mechanism to control this is required in production to avoid flooding the Image Repository with indecipherable output data.

Appendix A: ML System: Process description of ExecuteML

WPS 2.0 processs description XML file for ExecuteML

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessOfferings xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/2.0"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
http://schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering processVersion="1.0.0" jobControlOptions="sync-execute async-
execute" outputTransmission="value reference">
    <wps:Process>
      <ows:Title>Execute_MLDecisionTreeClassificationAlgorithm</ows:Title>
      <ows:Abstract>Execute decision tree classification based on Apache
Spark</ows:Abstract>

<ows:Identifier>org.n52.geoprocessing.project.testbed14.ml.Execute_MLDecisionTreeClass
ificationAlgorithm</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Source data</ows:Title>
        <ows:Identifier>source-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Input minOccurs="0" maxOccurs="1">
        <ows:Title>The model</ows:Title>
        <ows:Identifier>model</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Output>
        <ows:Title>Model quality indicators</ows:Title>
        <ows:Identifier>model-quality</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Classified image</ows:Title>
        <ows:Identifier>classified-image</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
          <ns:Format mimeType="application/wcs"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

Appendix B: ML System: Process description of RetrainML

WPS 2.0 processs description XML file for RetrainML

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessOfferings xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/2.0"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
http://schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering processVersion="1.0.0" jobControlOptions="sync-execute async-
execute" outputTransmission="value reference">
    <wps:Process>
      <ows:Title>Re-rain_MLDecisionTreeClassificationAlgorithm</ows:Title>
      <ows:Abstract>Re-train decision tree classification based on Apache
Spark</ows:Abstract>

<ows:Identifier>org.n52.geoprocessing.project.testbed14.ml.ReTrain_MLDecisionTreeClass
ificationAlgorithm</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Source data</ows:Title>
        <ows:Identifier>source-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Training data</ows:Title>
        <ows:Identifier>training-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
          <ns:Format mimeType="application/x-zipped-shp"/>
          <ns:Format mimeType="application/x-zipped-shp" encoding="base64"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/2.1.2/feature.xsd"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.2.1/base/feature.xsd"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Input minOccurs="0" maxOccurs="1">
        <ows:Title>Model</ows:Title>
        <ows:Abstract>The model to re-train</ows:Abstract>
        <ows:Identifier>input-model</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Input>
```

```
      <wps:Output>
        <ows:Title>Model</ows:Title>
        <ows:Abstract>The re-trained model</ows:Abstract>
        <ows:Identifier>re-trained-model</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Model quality indicators</ows:Title>
        <ows:Identifier>model-quality</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Classified image</ows:Title>
        <ows:Identifier>classified-image</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="image/png"/>
          <ns:Format mimeType="image/png" encoding="base64"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Classified features</ows:Title>
        <ows:Identifier>classified-features</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-zipped-shp"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
          <ns:Format mimeType="text/xml; subtype=gml/3.1.1" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
          <ns:Format mimeType="application/vnd.geo+json"/>
          <ns:Format mimeType="application/x-zipped-shp" encoding="base64"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

Appendix C: ML System: Process description of TrainML

WPS 2.0 processs description XML file for TrainML

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessOfferings xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/2.0"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
http://schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering processVersion="1.0.0" jobControlOptions="sync-execute async-
execute" outputTransmission="value reference">
    <wps:Process>
      <ows:Title>Train_MLDecisionTreeClassificationAlgorithm</ows:Title>
      <ows:Abstract>Train decision tree classification based on Apache
Spark</ows:Abstract>

<ows:Identifier>org.n52.geoprocessing.project.testbed14.ml.Train_MLDecisionTreeClassif
icationAlgorithm</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Source data</ows:Title>
        <ows:Identifier>source-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Training data</ows:Title>
        <ows:Identifier>training-data</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-geotiff"/>
          <ns:Format mimeType="application/x-zipped-shp"/>
          <ns:Format mimeType="application/x-zipped-shp" encoding="base64"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/2.1.2/feature.xsd"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.2.1/base/feature.xsd"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Input minOccurs="0" maxOccurs="1">
        <ows:Title>Initial model parameters</ows:Title>
        <ows:Identifier>initial-model-parameters</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/json"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Output>
```

```
        <ows:Title>The model</ows:Title>
        <ows:Identifier>model</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Model quality indicators</ows:Title>
        <ows:Identifier>model-quality</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/zip"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Classified image</ows:Title>
        <ows:Identifier>classified-image</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="image/png"/>
          <ns:Format mimeType="image/png" encoding="base64"/>
        </wps:ComplexData>
      </wps:Output>
      <wps:Output>
        <ows:Title>Classified features</ows:Title>
        <ows:Identifier>classified-features</ows:Identifier>
        <wps:ComplexData xmlns:ns="http://www.opengis.net/wps/2.0">
          <ns:Format default="true" mimeType="application/x-zipped-shp"/>
          <ns:Format mimeType="text/xml" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
          <ns:Format mimeType="text/xml; subtype=gml/3.1.1" schema=
"http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
          <ns:Format mimeType="application/vnd.geo+json"/>
          <ns:Format mimeType="application/x-zipped-shp" encoding="base64"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

Appendix D: CVM: Default JSON response

Default JSON response of the CVM


```
{
    "results": [
        {
            "id": "4d2fda7b59b1092b2b028f8ddb3d2e01",
            "uri": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/automobile",
            "type": [
                "individual",
                "concept"
            ],
            "primary": false,
            "prefix": "codelist",
            "version": "1.0",
            "label": "automobile",
            "description": "Definition: A motor vehicle generally with four wheels
that carries a small number of passengers.  Description: [None Specified]",
            "namespace": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/",
            "localName": "automobile",
            "qname": "codelist:automobile",
            "category": [
                "Individual",
                "additionalProperties"
            ]
        },
        {
            "id": "c595cd29703ee5d500de62bb3d9fe1ed",
            "uri": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/autoRickshaw",
            "type": [
                "individual",
                "concept"
            ],
            "primary": false,
            "prefix": "codelist",
            "version": "1.0",
            "label": "autoRickshaw",
            "description": "Definition: A motorized vehicle usually with three wheels
having a space to carry a few passengers or a small load of freight.  Description:
[None Specified]",
            "namespace": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/",
            "localName": "autoRickshaw",
            "qname": "codelist:autoRickshaw",
            "category": [
                "Individual",
                "additionalProperties"
            ]
        },
        {
```

```
            "id": "1950c9a8a53fafe317cc22c825f50125",
            "uri": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/barge",
            "type": [
                "individual",
                "concept"
            ],
            "primary": false,
            "prefix": "codelist",
            "version": "1.0",
            "label": "barge",
            "description": "Definition: A flat-bottomed freight-carrying watercraft
that is propelled (for example: towed or pushed) by a separate watercraft (for
example: a tug).  Description: [None Specified]",
            "namespace": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/",
            "localName": "barge",
            "qname": "codelist:barge",
            "category": [
                "Individual",
                "additionalProperties"
            ]
        }
    ],
    "page": {
        "size": 3,
        "number": 0,
        "totalElements": 26
    }
}
```

Appendix E: CVM: Full JSON response

*JSON response of the CVM using fields=**

https://ogctb14.usersmarts.com/terms?size=1&fields=*


```
{
    "results": [
        {
            "id": "4d2fda7b59b1092b2b028f8ddb3d2e01",
            "uri": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/automobile",
            "type": [
                "individual",
                "concept"
            ],
            "classes": [
                "http://api.nsgreg.nga.mil/codelist/ConveyanceType"
            ],
            "releaseId": "1a87b01b922c46a9a44bd1643b1134a5",
            "primary": false,
            "prefix": "codelist",
            "version": "1.0",
            "deprecated": false,
            "label": "automobile",
            "labelMap": {
                "en": "automobile"
            },
            "description": "Definition: A motor vehicle generally with four wheels
that carries a small number of passengers.  Description: [None Specified]",
            "descriptionMap": {
                "en": "Definition: A motor vehicle generally with four wheels that
carries a small number of passengers.  Description: [None Specified]"
            },
            "prefLabel": "Automobile (Conveyance Type Codelist)",
            "prefLabelMap": {
                "en": "Automobile (Conveyance Type Codelist)"
            },
            "namespace": "http://api.nsgreg.nga.mil/codelist/ConveyanceType/",
            "localName": "automobile",
            "languages": [
                "iso3:eng"
            ],
            "properties": [
                {
                    "property": {
                        "uri": "http://www.w3.org/2004/02/skos/core#topConceptOf",
                        "label": "topConceptOf"
                    },
                    "values": [
                        {
```

```
                            "uri":
"http://api.nsgreg.nga.mil/codelist/ConveyanceType_ConceptScheme",
                            "label": "ConveyanceType_ConceptScheme"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "http://www.w3.org/2004/02/skos/core#prefLabel",
                        "label": "prefLabel"
                    },
                    "values": [
                        {
                            "value": "Automobile (Conveyance Type Codelist)",
                            "lang": "en"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "http://www.w3.org/2004/02/skos/core#inScheme",
                        "label": "inScheme"
                    },
                    "values": [
                        {
                            "uri":
"http://api.nsgreg.nga.mil/codelist/ConveyanceType_ConceptScheme",
                            "label": "ConveyanceType_ConceptScheme"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "http://www.w3.org/2004/02/skos/core#definition",
                        "label": "definition"
                    },
                    "values": [
                        {
                            "value": "Definition: A motor vehicle generally with four
wheels that carries a small number of passengers.  Description: [None Specified]",
                            "lang": "en"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "
http://api.nsgreg.nga.mil/ontology/regx/1.0/itemStatus",
                        "label": "itemStatus"
                    },
                    "values": [
```

```
                        {
                            "uri":
"http://api.nsgreg.nga.mil/ontology/regx/1.0/ItemStatusType/valid",
                            "label": "valid"
                        }
                    ]
                },
                {
                    "property": {
                        "uri":
"http://api.nsgreg.nga.mil/ontology/regx/1.0/dateAccepted",
                        "label": "dateAccepted"
                    },
                    "values": [
                        {
                            "value": "2014-05-12T04:00:00Z"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "http://www.w3.org/2000/01/rdf-schema#label",
                        "label": "label"
                    },
                    "values": [
                        {
                            "value": "automobile",
                            "lang": "en"
                        }
                    ]
                },
                {
                    "property": {
                        "uri": "http://www.w3.org/2000/01/rdf-schema#isDefinedBy",
                        "label": "isDefinedBy"
                    },
                    "values": [
                        {
                            "uri": "http://nsgreg.nga.mil/ir/view?i=112883",
                            "label": "view?i=112883"
                        }
                    ]
                }
            ],
            "conceptScheme": {
                "uri":
"http://api.nsgreg.nga.mil/codelist/ConveyanceType_ConceptScheme",
                "label": "ConveyanceType_ConceptScheme"
            },
            "qname": "codelist:automobile",
            "current": true,
```

```
            "category": [
                "Individual",
                "additionalProperties"
            ],
            "isRoot": true
        }
    ],
    "page": {
        "size": 1,
        "number": 0,
        "totalElements": 26
    }
}
```

Appendix F: TB14 MoPoQ ML Task components

Please enter your TB14 MoPoQ components URLs into the following table:

Table 2. TB14 Components of the ML task

Components Type	Component Provider	Point of Contact	URL
D166-CVM	Image Matters	cullenr@imagemattersllc.com [mailto:cullenr@imagemattersllc.com]	https://ogctb14.usersmarts.com/terms
D132-Image Repository	52 North	b.pross@52north.org [mailto:b.pross@52north.org]	http://testbed.dev.52north.org:80/geoserver/
165-ML System (test client)	52 North	b.pross@52north.org [mailto:b.pross@52north.org]	http://testbed.dev.52north.org/tb14-d165-test-client/example.html
D164-KB API	Feng Chia University	will@gis.tw [mailto:will@gis.tw]	http://140.134.48.19/ML/
D141/D133-Client to ML KB and ML System	Envitia	Cameron.Brown@envitia.com [mailto:Cameron.Brown@envitia.com]	http://212.139.164.49/EnvitiaHorizon/
D030-Trained DL classifier	CRIM	mario.beaulieu@crim.ca [mailto:mario.beaulieu@crim.ca]	https://docker-registry.crim.ca/ogc-public/pytorch-classification:v1

Appendix G: Sample Pleiades Imagery

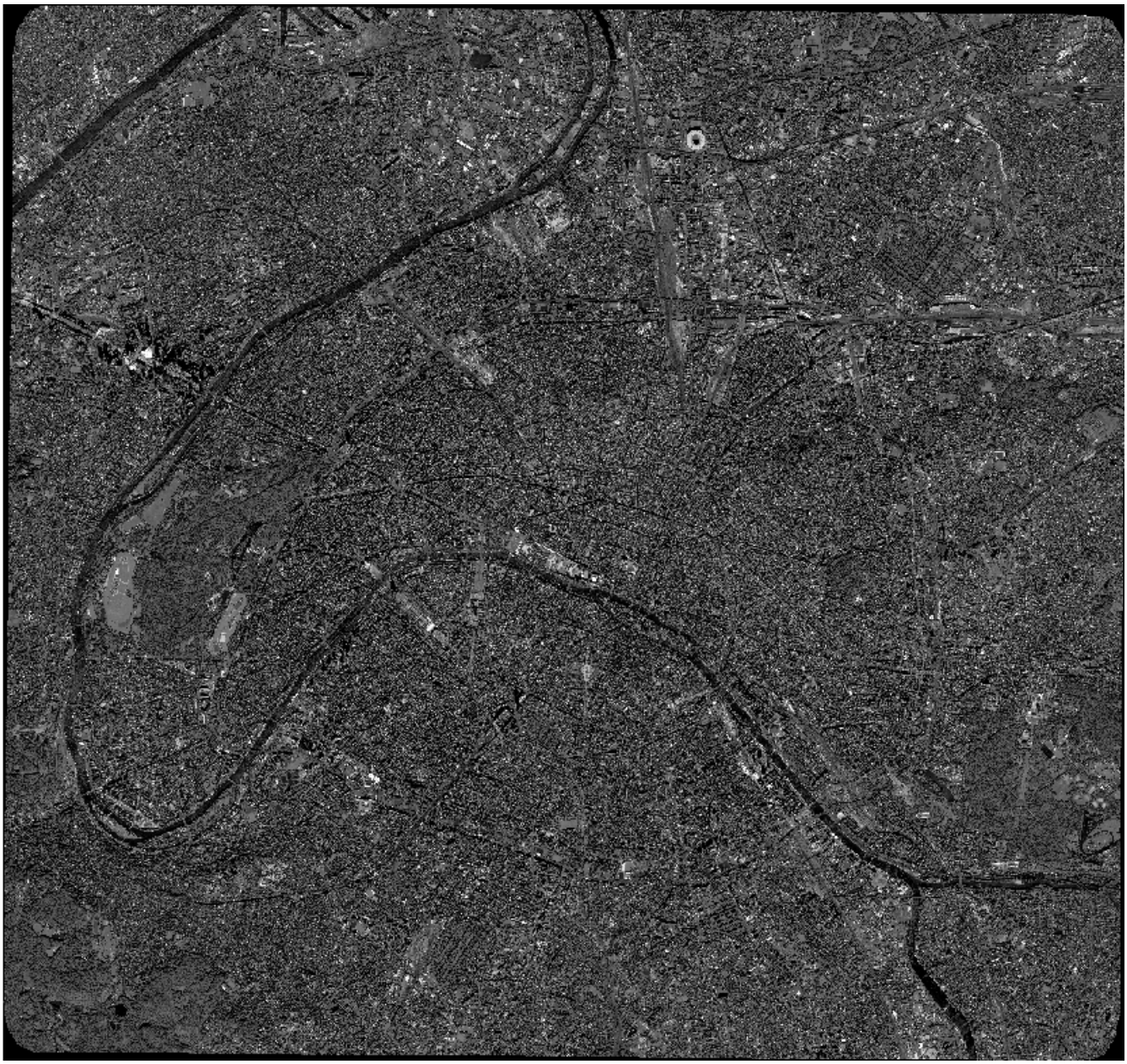

Figure 23. Pleiades Imagery of Paris, hosted in Image Repositoy and provided by EU SatCen

Appendix H: Revision History

Table 3. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
June 1, 2018	T. Landry	.1	all	initial version
July 11, 2018	T. Landry	.2	all	introduction of TOC and bibliography
September 8, 2018	T. Landry	.3	all	deliverables, recommendations, landscape
October 3, 2018	T. Landry	.4	all	summary, citations, typos
October 11, 2018	T. Landry	.5	all	future work
October 28, 2018	T. Landry	.6	all	annexes, integration of contributed content
October 31, 2018	T. Landry	.7	all	near-final DER
November 21, 2018	T. Landry	.8	all	additional comments and contributed edits, final DER
November 27, 2018	T. Landry	.9	POC, demo	missing screenshots

Appendix I: Bibliography

1. Yang, W., Whiteside, A.: OGC Discussion Paper: Web Image Classification Service (WICS) Implementation Specification. OGC 05-017,Open Geospatial Consortium, https://portal.opengeospatial.org/files/?artifact_id=8981 (2018).
2. Sacramento, P., others: OGC Testbed-14: Application Package ER. OGC 18-049,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
3. Ball, J., Anderson, D., Wei, P.: State-of-the-Art And Gaps For Deep Learning on Limited Training Data in Remote Sensing. International Geoscience and Remote Sensing Symposium (IGARSS). (2018).
4. Fu, J., Rui, Y.: Advances in deep learning approaches for image tagging. APSIPA Transactions on Signal and Information Processing. 6, e11 (2017).
5. Yuille, A.L., Liu, C.: Deep Nets: What have they ever done for Vision? CoRR. abs/1805.04025, (2018).
6. Conklin, B., Marchlevski, T., Pasechnik, T., Simmons, S., Sullivan, J., Walton, D., Young, J.: Bridging the Gap Between Analysts and Artificial Intelligence. USGIF 2018 State and Future of GEOINT Report. (2018).
7. Behm, D., Bryan, T., Lordemann, J., Thomas, S.R.: The Past, Present, and Future of Geospatial Data Use. USGIF 2018 State and Future of GEOINT Report. (2018).
8. Brian Collins, I., Heyman, O., Ramírez, J., King, T., Schmidt, B., Young, P.M., Kroll, K.C., Driverand, R., Niedner, C.: Modeling Outcome-Based Geospatial Intelligence. USGIF 2018 State and Future of GEOINT Report. (2018).
9. Parrett, C.M., Crooks, A., Pike, T.: The Future of GEOINT: Data Science Will Not Be Enough. USGIF 2018 State and Future of GEOINT Report. (2018).
10. Bacastow, T.M., Brown, A., Chang, G., Lindenbaum, D.G.D.: Actionable Automation: Assessing the Mission-Relevance of Machine Learning for the GEOINT Community. USGIF 2018 State and Future of GEOINT Report. (2018).
11. Sarojak, M., Kepner, D., Tracy, R., Robinson, C.A., Gruber, C., Feldman, D.: An Orchestra of Machine Intelligence. USGIF 2018 State and Future of GEOINT Report. (2018).
12. Kang, J., Korner, M., Wang, Y., Taubenbock, H., Zhu, X.X.: Building instance classification using street view images. ISPRS Journal of Photogrammetry and Remote Sensing. (2018).
13. Scott, G.J., England, M.R., Starms, W.A., Marcum, R.A., Davis, C.H.: Training Deep Convolutional Neural Networks for Land Cover Classification of High-Resolution Imagery. IEEE Geoscience and Remote Sensing Letters. 14, 549–553 (2017).
14. Vetrivel, A., Gerke, M., Kerle, N., Nex, F., Vosselman, G.: Disaster damage detection through synergistic use of deep learning and 3D point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning. ISPRS Journal of Photogrammetry and Remote Sensing. 140, 45–59 (2018).
15. Deng, X., Zhu, Y., Newsam, S.: What Is It Like Down There? Generating Dense Ground-Level Views and Image Features From Overhead Imagery Using Conditional Generative Adversarial Networks. ArXiv e-prints. (2018).

16. Sghaier, M.O., Hammami, I., Foucher, S., Lepage, R.: Flood hazard mapping from SAR images using texture analysis and fuzzy logic. 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). 1900–1903 (2017).

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. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2018).

18. Karpatne, A., Ebert-Uphoff, I., Ravela, S., Babaie, H.A., Kumar, V.: Machine Learning for the Geosciences: Challenges and Opportunities. CoRR. abs/1711.04708, (2017).

19. Kuo, K.-S., Pan, Y., Zhu, F., Wang, J., Rilee, M., Yu, H.: A Big Earth Data Platform Exploiting Transparent Multimodal Parallelization. International Geoscience and Remote Sensing Symposium (IGARSS). (2018).

20. Redmon, J., Farhadi, A.: YOLO9000: Better, Faster, Stronger. CoRR. abs/1612.08242, (2016).

21. Zhou, B., Khosla, A., Lapedriza, Àgata, Torralba, A., Oliva, A.: Places: An Image Database for Deep Scene Understanding. CoRR. abs/1610.02055, (2016).

22. Dahmane, M., Foucher, S., Beaulieu, M., Riendeau, F., Bouroubi, Y., Benoit, M.: Object detection in pleiades images using deep features. 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). 1552–1555 (2016).

23. Xu, N., Price, B.L., Cohen, S., Yang, J., Huang, T.S.: Deep Interactive Object Selection. CoRR. abs/1603.04042, (2016).

24. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The Cityscapes Dataset for Semantic Urban Scene Understanding. CoRR. abs/1604.01685, (2016).

25. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. IEEE Transactions on Pattern Analysis and Machine Intelligence. 40, 834–848 (2018).

26. Holmes, C., Tucker, C., Tuttle, B.: GEOINT at Platform Scale. USGIF 2018 State and Future of GEOINT Report. (2018).

27. Lin, T.-Y., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common Objects in Context. CoRR. abs/1405.0312, (2014).

28. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Li, F.-F.: ImageNet Large Scale Visual Recognition Challenge. CoRR. abs/1409.0575, (2014).

29. Hinton, G.E., Osindero, S., Teh, Y.-W.: A Fast Learning Algorithm for Deep Belief Nets. Neural Comput. 18, 1527–1554 (2006).

30. F. Elsayed, G., Goodfellow, I., Sohl-Dickstein, J.: Adversarial Reprogramming of Neural Networks, https://arxiv.org/abs/1806.11146, (2018).

31. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature. 521, 436 (2015).

32. Sacramento, P., others: OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report. OGC 18-050,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2005).