July 3, 2019

Acceptance Test Plan

Version 1.0.0

Members
Andrei Danila | 1034559
Bogdan Enache | 1035066
Gergana Goncheva | 1037010
Loïc Alexander Hijl | 1002745
Adrian-Stefan Mares | 0993873
Veselin Minev | 1014541
Thanh-Dat Nguyen | 1036672
Antoine Labasse Lutou Nijhuis | 1016657
Claudiu-Teodor Nohai | 1038442
Dragos Mihai Serban | 1033859
Tsvetan Zahariev | 1035269
Sonya Zarkova | 1034611

Project managers
Yuxuan Zhang
Stefan Tanja

Supervisor
Erik Luit
Ion Barosan

Customer
Joaquin Vanschoren

Abstract

This document contains the acceptance test plan for the Deep Learning (DL) Extension Library for OpenML, which is developed by the OpenML Support Squad. The tests contained in this Acceptance Test Plan (ATP) reflect the requirements specified in the User Requirements Document (URD) [1]. This document complies with the European Space Agency (ESA) software standards [2].

Contents

1 Introduction **6**
1.1 Purpose 6
1.2 Definitions and Abbreviations 6
1.2.1 Definitions 6
1.2.2 Abbreviations 7
1.3 List of references 8
1.4 Overview 8

2 Test plan **9**
2.1 Test items 9
2.2 Features to be tested 9
2.3 Test deliverables 10
2.4 Testing tasks 10
2.5 Environmental needs 10
2.6 Test case pass/fail criteria 11

3 Test case specification **12**
3.1 Test structure 12
3.2 Converting to OpenML Flow 12
3.2.1 ATA - 01 12
3.2.2 ATA - 02 13
3.2.3 ATA - 03 13
3.2.4 ATA - 04 14
3.2.5 ATA - 05 14
3.2.6 ATA - 06 15
3.2.7 ATA - 07 16
3.3 Converting from OpenML Flow 16
3.3.1 ATB - 01 16
3.3.2 ATB - 02 17
3.3.3 ATB - 03 18
3.3.4 ATB - 04 18
3.4 Data reliability 19
3.4.1 ATC - 01 19
3.4.2 ATC - 02 20
3.4.3 ATC - 03 21
3.4.4 ATC - 04 22
3.5 General constraints 23
3.5.1 ATD - 01 23
3.5.2 ATD - 02 23
3.5.3 ATD - 03 24
3.5.4 ATD - 04 25

3.6 Visualization . . . 26
3.6.1 ATE - 01 . . . 26
3.6.2 ATE - 02 . . . 27
3.6.3 ATE - 03 . . . 28

4 Test procedure 29
4.1 KerasExtension test procedure . . . 29
4.1.1 Test procedure identifier . . . 29
4.1.2 Purpose . . . 29
4.1.3 Procedure steps . . . 29
4.2 PytorchExtension test procedure . . . 30
4.2.1 Test procedure identifier . . . 30
4.2.2 Purpose . . . 30
4.2.3 Procedure steps . . . 30
4.3 MXNetExtension test procedure . . . 30
4.3.1 Test procedure identifier . . . 30
4.3.2 Purpose . . . 31
4.3.3 Procedure steps . . . 31
4.4 ONNXExtension test procedure . . . 31
4.4.1 Test procedure identifier . . . 31
4.4.2 Purpose . . . 31
4.4.3 Procedure steps . . . 31
4.5 Visualization . . . 32
4.5.1 Test procedure identifier . . . 32
4.5.2 Purpose . . . 32
4.5.3 Procedure steps . . . 32

5 Test reports 33

Document Status Sheet

Document Title: Acceptance Test Plan
Identification: ATP/1.0.0
Version: 1.0.0
Authors: A.Danila, B.Enache, G.Goncheva, L.A.Hijl, A.Mares, V.Minev, T.Nguyen, A.L.L.Nijhuis, C.Nohai, D.M.Serban, T.Zahariev, S.Zarkova

Document History

Version	Date	Author(s)	Summary
0.0.1	25-04-2019	T. Zahariev	Created initial document
0.0.2	25-04-2019	T. Zahariev & L.A. Hijl	Added macros and glossary
0.0.3	25-04-2019	T. Zahariev & L.A. Hijl	Added introduction layout
0.0.4	25-04-2019	T. Zahariev & L.A. Hijl	Added layout
0.0.5	01-05-2019	L.A. Hijl	Added chapters
0.0.6	27-05-2019	S. Zarkova	Added the Abstract
0.0.7	28-05-2019	S.Zarkova	Added the Introduction
0.0.8	17-06-2019	All authors	Completed first draft version
0.0.9	19-06-2019	All authors	Implemented internal feedback
0.1.0	20-06-2019	All authors	Fixed supervisor feedback
1.0.0	21-06-2019	All authors	Final edit

1 | Introduction

1.1 Purpose

This document contains the acceptance test plan and results for project "Sharing deep learning models", which represents an extension for the OpenML platform. It reflects on the user requirements found in Section 3 of the User Requirements Document (URD) [1]. The client - Joaquin Vanschoren, needs to approve the tests before they are executed. Consequently, he will have to perform the tests himself, following the described procedures, after which the results will be reported in this document. The DL Extension Library can only be accepted by the client after this acceptance test plan has been completed successfully, i.e. every test's outcome matches the expected one .
This ATP is written by the OpenML Support Squad in accordance to Joaquin Vanschoren's preferences. Any further changes to this document requires the full consent of both parties, aforementioned.

1.2 Definitions and Abbreviations

1.2.1 Definitions

DL Extension Library	The product to be created in order to convert the deep learning models and the model transfer specifications (i.e. Open Neural Network eXchange (ONNX) and MLflow) into a format supported by the OpenML platform.
OpenML Python API	"A connector to the collaborative machine learning platform OpenML.org. The OpenML Python package allows to use datasets and tasks from OpenML together with scikit-learn and share the results online" [3].
OpenML flow	The representation of machine learning models in the OpenML platform.
Apache MXNet	"Apache MXNet is an open-source deep learning software framework, used to train, and deploy deep neural networks. It is scalable, allowing for fast model training, and supports multiple programming languages (C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, and Wolfram Language)" [4].
Correct Conversion	The conversion is correct when the converted output can be converted back to the corresponding original input (the information is preserved).
Dataset	"Datasets are pretty straight-forward. They simply consist of a number of rows, also called instances, usually in tabular form" [3].

Deep learning model	A neural network with more than three layers. Deep learning is a subset of machine learning.
Fold	A subset of the training data.
Hyperparameter	A hyperparameter is a parameter of a machine learning model that is set before training the model. Such parameters can be layers, seeds, etc.
Joaquin Vanschoren	Client of this project. Founder of OpenML.
Keras	"Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano" [5].
Neural Network	"Computing systems vaguely inspired by the biological neural networks that constitute animal brains." It is "a framework for many different machine learning algorithms" [6].
OpenML	An online machine learning platform for sharing and organizing data, machine learning algorithms and data experiments.
OpenML Support Squad	Name of the development team.
Project "Sharing deep learning models"	The project on which the OpenML Support Squad is working on.
PyTorch	"An open source deep learning platform that provides a seamless path from research prototyping to product deployment" [7].
Run	"A run is a particular flow, that is algorithm, with a particular parameter setting, applied to a particular task" [3].
Task	"A task consists of a dataset, together with a machine learning task to perform, such as classification or clustering and an evaluation method. For supervised tasks, this also specifies the target column in the data" [3].

1.2.2 Abbreviations

API	Application Programming Interface.
AT	Acceptance Testing.
ATP	Acceptance Test Plan.
ATR	Acceptance Test Report.
DL	Deep Learning.
ESA	European Space Agency.
MXNet	Apache MXNet.
ONNX	Open Neural Network eXchange.
SRD	Software Requirements Document.
URD	User Requirements Document.
UTP	Unit Test Plan.

1.3 List of references

[1] OpenML Support Squad. *DL Extension Library, URD User Requirement Document version 1.0.0.*

[2] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[3] OpenML. *OpenML Documentation.* URL: `https://docs.openml.org/` (visited on 04/26/2019).

[4] Wikipedia. *Apache MXNet.* URL: `https://en.wikipedia.org/wiki/Apache_MXNet` (visited on 04/26/2019).

[5] Keras. *Keras homepage.* URL: `https://keras.io/` (visited on 04/26/2019).

[6] Wikipedia. *Artificial Neural Network.* URL: `https://en.wikipedia.org/wiki/Artificial_neural_network` (visited on 04/30/2019).

[7] PyTorch. *PyTorch homepage.* URL: `https://pytorch.org/` (visited on 09/05/2019).

1.4 Overview

The remainder of this ATP document contains four additional chapters. First in order is chapter 2 - "Test Plan" with the corresponding Sections: Section 2.1 containing the test items, Section 2.2 with the features to be tested, Section 2.3 describing the test deliverables, Section 2.4 with the concrete testing tasks, Section 2.5 outlining the environmental needs and lastly Section 2.6 with the definition for the pass / fail criteria of tests.
Chapter 3 - "Test Case Specification" follows, containing a Section for each functionality of the DL Extension Library and the visualization module. The sections constitute of uniquely identifiable test cases for the different libraries supported. Moreover, one can find the items to be tested, the input specification for each test case, the information in regard output specifications, and lastly the test environmental needs.
Next is chapter 4 - "Test Procedure". Each test procedure contains descriptions in regard to the purpose of it, the test cases this procedure executes and its process (i.e. how to log, set up, start, etc). There are four test procedures for the deep learning libraries and specifications supported, as well as one concerning the visualization module.
The document finishes with chapter 5 - "Test reports" Acceptance Test Report (ATR), depicting the test runs, identification of the test procedure, the time when the test was performed and by whom. For each test case in the procedure it is determined whether it passes.

2 | Test plan

2.1 Test items

The software to be tested is the DL Extension Library. It expands on the already existing Python extension API used by OpenML to accommodate for Deep Learning models. Testing of the DL Extension Library entails testing whether the desired OpenML procedures on models for all Deep learning libraries covered by the DL Extension Library are functioning correctly.

2.2 Features to be tested

The features to be tested consist of all implemented features of the DL Extension Library. Passing the acceptance test entails passing all tests for serialization, deserialization and running on a fold of the models built on the following Deep Learning libraries or specifications: Keras, PyTorch, ONNX and Apache MXNet (MXNet). All of which are described by the "*must have* functional requirements" chapter 3 of the Software Requirements Document (SRD).

Additionally, visualization is implemented in a completely separate module from the DL Extension Library and also has to be tested. Only implemented features will be tested during the Acceptance Test.

The DL Extension Library is part of the OpenML Python API that will be imported by the users in their Python project. The OpenML Python API handles all interactions with the OpenML platform. Whenever necessary, the OpenML Python API calls the DL Extension Library in order to convert the deep learning model to an OpenML flow, the format supported by the OpenML platform. The OpenML Support Squad is not responsible for the development and maintenance of the OpenML Python API. The OpenML Support Squad has not modified the OpenML Python API since it is outside of the scope of the project "Sharing deep learning models". Because of this, the OpenML Support Squad will not create acceptance tests for functions and use cases that do not make use of the DL Extension Library.

The contribution of OpenML Support Squad consists of two parts: creating the DL Extension Library and the visualization module.

The DL Extension Library was built to provide support for the following deep learning libraries and specifications on the OpenML platform: Keras, PyTorch, MXNet and ONNX. These libraries have already been created by their respective developers and, as such, are not part of this project and are not under the responsibility of OpenML Support Squad. Because of this, the OpenML Support Squad will not create acceptance tests for the functions that belong to these frameworks.

The visualization module was built to serve as a prototype which can be integrated into the website. It allows users to visualize their models and model results.

2.3 Test deliverables

Mandatory deliverables before testing:

- URD
- Unit Test Plan (UTP) sections 1 through 4
- ATP sections 1 through 4
- Acceptance test input
- Finalized version of the DL Extension Library

Mandatory deliverables after testing has concluded:

- UTP section 5
- Acceptance test output data
- Problem reports

2.4 Testing tasks

Before the AT phase can start, the following tasks have to be completed:

- Designing of acceptance tests
- Linking test cases to user requirements in the URD
- Creation of Acceptance Testing (AT) input data
- Ensuring all necessary environmental needs are met

Only once these tasks are completed, an AT can be performed. This should be done following the procedures described in this document.

2.5 Environmental needs

To perform acceptance testing on the DL Extension Library, the following environmental requirements need to be satisfied:

- A system capable of utilizing the DL Extension Library requires at least the following specifications:

CPU	$\geq$ 2.0 GHz x64 or equivalent
RAM	$\geq$ 8GB
Disk space	$\geq$ 10GB
Operating system	Windows 7 or later, macOS, or Linux

- Python version 3.5, 3.6 or 3.7
- Internet connection needed for downloading and uploading OpenML flows as well as publishing results on runs

2.6 Test case pass/fail criteria

In the next chapter, we describe the individual test cases. Their pass conditions are described in the corresponding contracts (expected outcome). If the output of an acceptance test differs from the contract, the test fails. The acceptance test plan as a whole passes only if every individual acceptance test passes.

3 | Test case specification

3.1 Test structure

Test case identifier: The name and number of the test
Description: A brief description of the test
Precondition: Conditions needed to be fulfilled before the test can begin
Requirements fulfilled: The URD requirements that need to be fulfilled
Input specification: Input parameters used for the test
Output specification: Expected result of each function

3.2 Converting to OpenML Flow

All the tests from this section are specific to conversion to OpenML flow.

3.2.1 ATA - 01

Description: Run and publish a Keras model on a classification task.
Precondition: Have a Keras model respecting the general constraints from section 3.5, a classification task, and the Keras library, the OpenML library and the Keras extension installed.
Requirements fulfilled: URF - 03; URF - 17; URF - 20; URC - 37.

Input specification:	***Output specification:***
1. Import only `keras` and the `openml.extensions.keras`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the classification task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the OpenML website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.2.2 ATA - 02

Description: Run and publish a Keras model on a regression task.
Precondition: Have a Keras model respecting the general constraints from section 3.5, a regression task, and the Keras library, the OpenML library and the Keras extension installed.
Requirements fulfilled: URF - 03; URF - 17; URF - 21; URC - 37.

Input specification:	***Output specification:***
1. Import only `keras` and the `openml.extensions.keras`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the OpenML website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.2.3 ATA - 03

Description: Run and publish a PyTorch model on a classification task.
Precondition: Have a PyTorch model respecting the general constraints from section 3.5, a classification task, and the PyTorch library, the OpenML library and the PyTorch extension installed.
Requirements fulfilled: URF - 05; URF - 17; URF - 22; URC - 38.

Input specification:	***Output specification:***
1. Import only `pytorch` and the `openml.extensions.pytorch`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	

3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.2.4 ATA - 04

Description: Run and publish a PyTorch model on a regression task.
Precondition: Have a PyTorch model respecting the general constraints from section 3.5, a regression task, and the PyTorch library, the OpenML library and the PyTorch extension installed.
Requirements fulfilled: URF - 05; URF - 17; URF - 23; URC - 38.

Input specification:	***Output specification:***
1. Import only `pytorch` and the `openml.extensions.pytorch`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.2.5 ATA - 05

Description: Run and publish a MXNet model on a classification task.
Precondition: Have a MXNet model respecting the general constraints from section 3.5, a clas-

sification task, and the MXNet library, the OpenML library and the MXNet extension installed.
Requirements fulfilled: URF - 09; URF - 17; URF - 24; URC - 43.

Input specification:	***Output specification:***
1. Import only `mxnet` and the `openml.extensions.mxnet`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.2.6 ATA - 06

Description: Run and publish an MXNet model on a regression task.
Precondition: Have a MXNet model respecting the general constraints from section 3.5, a regression task, and the MXNet library, the OpenML library and the MXNet extension installed.
Requirements fulfilled: URF - 09; URF - 17; URF - 25; URC - 43.

Input specification:	***Output specification:***
1. Import only `mxnet` and the `openml.extensions.mxnet`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Go on the website to the flow section.	
7. Confirm that the flow exists.	

8. View the hyperparameter representation.	

3.2.7 ATA - 07

Description: Run and publish an ONNX specification on a task.
Precondition: Have an ONNX specification respecting the general constraints from section 3.5, a task, and the ONNX library, the OpenML library and the ONNX extension installed.
Requirements fulfilled: URF - 01; URF - 17; URC - 39.

Input specification:	***Output specification:***
1. Import only `onnx` and the `openml.extensions.onnx`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the specification by calling `openml.runs.run_model_on_task(specification, tasks, avoid_duplicate_runs = False)`.	
4. Publish the specification by calling `publish()`.	
	5. The specification is uploaded on the website under the OpenML flow format.
6. Go on the website to the flow section.	
7. Confirm that the flow exists.	
8. View the hyperparameter representation.	

3.3 Converting from OpenML Flow

All the tests from this section are specific to conversion from OpenML flow.

3.3.1 ATB - 01

Description: Download and run an OpenML flow representing a Keras model.
Precondition: Have an OpenML flow representing a Keras model respecting the general constraints from section 3.5 on the OpenML website, a task, and the Keras library, the OpenML library and the Keras extension installed.

Requirements fulfilled: URF - 04; URC - 40.

Input specification:	***Output specification:***
1. Import only `keras` and the `openml.extensions.keras`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Get the flow by calling `openml.flows.get_flow(id, reinstantiate = True)`, where `id` is the identification number of the flow on OpenML.	
4. Run the model by calling `openml.runs.run_model_on_task(flow.model, tasks, avoid_duplicate_runs = False), where flow is the downloaded OpenML flow.`	
	5. Process finished with a certain exit code.
6. Confirm that exit code is 0 (i.e. process finished without an error).	

3.3.2 ATB - 02

Description: Download and run an OpenML flow representing a PyTorch model.
Precondition: Have an OpenML flow representing a PyTorch model respecting the general constraints from section 3.5 on the OpenML website, a task and the PyTorch library, the OpenML library and the PyTorch extension installed.
Requirements fulfilled: URF - 06; URC - 41.

Input specification:	***Output specification:***
1. Import only `pytorch` and the `openml.extensions.pytorch`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Get the flow by calling `openml.flows.get_flow(id, reinstantiate = True)`, where `id` is the identification number of the flow on OpenML.	

4. Run the model by calling `openml.runs.run_model_on_task(flow.model, tasks, avoid_duplicate_runs = False)`, where `flow` is the downloaded OpenML flow.

5. Process finished with a certain exit code.

6. Confirm that exit code is 0 (i.e. process finished without an error).

3.3.3 ATB - 03

Description: Download and run an OpenML flow representing a MXNet model.
Precondition: Have an OpenML flow representing a MXNet model respecting the general constraints from section 3.5 on the OpenML website, a task, and the MXNet library, the OpenML library and the MXNet extension installed.
Requirements fulfilled: URF - 10; URC - 45.

Input specification:

1. Import only `mxnet` and the `openml.extensions.mxnet`.

2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.

3. Get the flow by calling `openml.flows.get_flow(id, reinstantiate = True)`, where `id` is the identification number of the flow on OpenML.

4. Run the model by calling `openml.runs.run_model_on_task(flow.model, tasks, avoid_duplicate_runs = False)`, where `flow` is the downloaded OpenML flow.

6. Confirm that exit code is 0 (i.e. process finished without an error).

Output specification:

5. Process finished with a certain exit code.

3.3.4 ATB - 04

Description: Download and run an OpenML flow representing an ONNX specification.
Precondition: Have an OpenML flow representing an ONNX specification respecting the general constraints from section 3.5 on the OpenML website, a task, and the ONNX library, the

OpenML library and the ONNX extension installed.
Requirements fulfilled: URF - 02; URC - 42.

Input specification:

1. Import only `onnx` and the `openml.extensions.onnx`.

2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.

3. Get the flow by calling `openml.flows.get_flow(id, reinstantiate = True)`, where `id` is the identification number of the flow on OpenML.

4. Run the specification by calling `openml.runs.run_model_on_task(flow.model, tasks, avoid_duplicate_runs = False), where flow is the downloaded OpenML flow`. NOTE: flow.model represents the specification of the flow, not the actual model of the flow. This is because of how the platform was built initially by other people.

Output specification:

5. Process finished with a certain exit code.

6. Confirm that exit code is 0 (i.e. process finished without an error).

3.4 Data reliability

3.4.1 ATC - 01

Description: Check if the DL Extension Library correctly converts a Keras model.
Precondition: Have a Keras model respecting the general constraints from section 3.5, a task, and the Keras library, the OpenML library and the Keras extension installed.
Requirements fulfilled: URC - 18; URC - 20.

Input specification:

1. Import only `keras` and the `openml.extensions.keras`.

2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.

Output specification:

3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Get the `flow_id` from the run. This id corresponds to the id of the model that has just been published on the website.	
7. Get the flow by calling `openml.flows.get_flow(flow_id, reinstantiate = True)`.	
8. Compare the configuration of the initial model `initial_model.get_config()` with the one of the new model `flow.model.get_config()` by using ==.	
	9. The comparison returns a certain value.
10. Confirm that the value is `True`.	

3.4.2 ATC - 02

Description: Check if the DL Extension Library correctly converts a PyTorch model.
Precondition: Have a PyTorch model respecting the general constraints from section 3.5, a task, and the PyTorch library, the OpenML library and the PyTorch extension installed.
Requirements fulfilled: URC - 19; URC - 21.

Input specification:	***Output specification:***
1. Import only `pytorch` and the `openml.extensions.pytorch`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the model by calling `publish()`.	
	5. The model is uploaded on the website under the OpenML flow format.
6. Get the `flow_id` from the run. This id corresponds to the id of the model that has just been published on the website.	

7. Get the flow by calling `openml.flows.get_flow(flow_id, reinstantiate = True)`.

8. Compare the string conversion of the initial model `initial_model` with the one of the new model `flow.model` by using ==.

9. The comparison returns a certain value.

10. Confirm that the value is `True`.

3.4.3 ATC - 03

Description: Check if the DL Extension Library correctly converts a MXNet model.
Precondition: Have a MXNet model respecting the general constraints from section 3.5, a task, and the MXNet library, the OpenML library and the MXNet extension installed.
Requirements fulfilled: URC - 27; URC - 29.

Input specification:

Output specification:

1. Import only `mxnet` and the `openml.extensions.mxnet`.

2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.

3. Run the model by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.

4. Publish the model by calling `publish()`.

5. The model is uploaded on the website under the OpenML flow format.

6. Get the `flow_id` from the run. This id corresponds to the id of the model that has just been published on the website.

7. Get the flow by calling `openml.flows.get_flow(flow_id, reinstantiate = True)`.

8. Compare the string conversion of the initial model `(initial_model) with the one of the new model flow.model by using ==`.

9. The comparison returns a certain value.

10. Confirm that the value is `True`.

3.4.4 ATC - 04

Description: Check if the DL Extension Library correctly converts an ONNX specification.
Precondition: Have an ONNX model respecting the general constraints from section 3.5, a task, and the ONNX library, the OpenML library and the ONNX extension installed.
Requirements fulfilled: URC - 22; URC - 23.

Input specification:	***Output specification:***
1. Import only `onnx` and the `openml.extensions.onnx`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Run the specification by calling `openml.runs.run_model_on_task(model, tasks, avoid_duplicate_runs = False)`.	
4. Publish the specification by calling `publish()`.	
	5. The specification is uploaded on the website under the OpenML flow format.
6. Get the `flow_id` from the run. This id corresponds to the id of the specification that has just been published on the website.	
7. Get the flow by calling `openml.flows.get_flow(flow_id, reinstantiate = True)`.	
8. Set the weights and biases of the original specification to 0, order the nodes in the graph, since the flow does not store weights or biases and the graph is sorted for OpenML.	
9. Compare the string representation of the initial specification `initial_specification` with the one of the new specification `flow.model` by using ==.	
	10. The comparison returns a certain value.
11. Confirm that the value is `True`.	

3.5 General constraints

3.5.1 ATD - 01

Description: Check if the DL Extension Library correctly converts on python 3.5/3.6/3.7 a Keras model of up to 1GB in size and up to 30 layers.
Precondition: Have python 3.5/3.6/3.7, the Keras library, the OpenML library and the Keras extension already installed.
Requirements fulfilled: URC - 01, URC - 09, URC - 34.

Input specification:	***Output specification:***
1. Import only `keras` and the `openml.extensions.keras`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Create a Keras model with up to 30 layers.	
4. Save the Keras model on your machine by using the command `keras.models.save_model(yourModel, yourPath)`.	
	5. The model is saved as a file to the path specified.
6. Confirm that the size of the file is no more than 1GB.	
7. Run the model by calling `openml.runs.run_model_on_task(yourModel, tasks, avoid_duplicate_runs = False)`.	
8. Publish the model by calling `publish()`.	
	9. The model is uploaded on the website under the OpenML flow format.
10. Go on the website to the flow section.	
11. Confirm that the flow exists.	

3.5.2 ATD - 02

Description: Check if the DL Extension Library correctly converts on python 3.5/3.6/3.7 a PyTorch model of up to 1GB in size and up to 30 layers.
Precondition: Have python 3.5/3.6/3.7, the PyTorch library, the OpenML library and the PyTorch extension already installed.
Requirements fulfilled: URC - 02, URC - 10, URC - 34.

Input specification:	***Output specification:***
1. Import only `pytorch` and the `openml.extensions.pytorch`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	
3. Create a PyTorch model with up to 30 layers.	
4. Save the PyTorch model on your machine by using the command `torch.save(yourModel, yourPath)`.	
	5. The model is saved as a file to the path specified.
6. Confirm that the size of the file is no more than 1GB.	
7. Run the model by calling `openml.runs.run_model_on_task(yourModel, tasks, avoid_duplicate_runs = False)`.	
8. Publish the model by calling `publish()`.	
	9. The model is uploaded on the website under the OpenML flow format.
10. Go on the website to the flow section.	
11. Confirm that the flow exists.	

3.5.3 ATD - 03

Description: Check if the DL Extension Library correctly converts on python 3.5/3.6/3.7 a MXNet model of up to 1GB in size and up to 30 layers.
Precondition: Have python 3.5/3.6/3.7, the MXNet library, the OpenML library and the MXNet extension already installed.
Requirements fulfilled: URC - 06, URC - 14, URC - 34.

Input specification:	***Output specification:***
1. Import only `mxnet` and the `openml.extensions.mxnet`.	
2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.	

3. Create a MXNet model with up to 30 layers.

4. Save the MXNet model on your machine by using the command `yourModel.export(file, epoch=1)`..

5. The model is saved as a file to the path specified.

6. Confirm that the size of the file is no more than 1GB.

7. Run the model by calling `openml.runs.run_model_on_task(yourModel, tasks, avoid_duplicate_runs = False)`.

8. Publish the model by calling `publish()`.

9. The model is uploaded on the website under the OpenML flow format.

10. Go on the website to the flow section.

11. Confirm that the flow exists.

3.5.4 ATD - 04

Description: Check if the DL Extension Library correctly converts on python 3.5/3.6/3.7 an ONNX specification of up to 1GB in size and up to 30 layers.
Precondition: Have python 3.5/3.6/3.7, the ONNX library, the OpenML library and the ONNX extension already installed.
Requirements fulfilled: URC - 03, URC - 11, URC - 34.

Input specification:

1. Import only `onnx` and the `openml.extensions.onnx`.

2. Get the task by calling `openml.tasks.get_task(id)`, where `id` is the identification number of the task on OpenML.

3. Create a MXNet model with up to 30 layers and prepare for ONNX conversion by saving the sym and params with `sym_model.save(symPath)` and `model.save_params(paramsPath)`.

4. Save the ONNX specification on your machine by using the command `onnx.save(yourSpecification, yourPath)`.

Output specification:

5. The specification is saved as a file to the path specified.

6. Confirm that the size of the file is no more than 1GB.

7. Run the specification by calling `openml. runs.run_model_on_task( yourSpecification, tasks, avoid_duplicate_runs = False)`.

8. Publish the specification by calling `publish()`.

9. The specification is uploaded on the website under the OpenML flow format.

10. Go on the website to the flow section.

11. Confirm that the flow exists.

3.6 Visualization

3.6.1 ATE - 01

Description: Check the learning curve graphs of a classification task.
Precondition: A run with its associated training data exists on the OpenML website. The visualization package is already installed. The OpenML library is already installed.
Requirements fulfilled: URF - 18.

Input specification:	***Output specification:***
1. Run the `visualize.py` file from the OpenML library.	
	2. The loading process finishes and outputs in the console the IP and port number of the local Dash website, possibly with a delay of a few seconds.
3. Open the local website in a browser.	
4. Find the id of the run on the OpenML website.	
5. Input the id in the run field of the local website and load it.	
	6. The loss graph of the run is rendered.
7. Check the loss graph.	
8. Choose the accuracy metric from the top-down menu.	

	9. The accuracy graph of the run is rendered.
10. Check the accuracy graph.	

3.6.2 ATE - 02

Description: Check the learning curve graphs of a regression task.
Precondition: A run with its associated training data exists on the OpenML website. The visualization package is already installed. The OpenML library is already installed.
Requirements fulfilled: URF - 18.

Input specification:	***Output specification:***
1. Run the `visualize.py` file from the OpenML library.	
	2. The loading process finishes and outputs in the console the IP and port number of the local Dash website, possibly with a delay of a few seconds.
3. Open the local website in a browser.	
4. Find the id of the run on the OpenML website.	
5. Input the id in the run field of the local website and load it.	
	6. The loss graph of the run is rendered.
7. Check the loss graph.	
8. Choose the mean square error metric from the top-down menu.	
	9. The mean square error graph of the run is rendered.
10. Check the mean square error graph.	
11. Choose the mean absolute error metric from the top-down menu.	
	12. The mean absolute error graph of the run is rendered.
13. Check the mean absolute error graph.	
14. Choose the root mean square error metric from the top-down menu.	
	15. The root mean square error graph of the run is rendered.
16. Check the root mean square error graph.	

3.6.3 ATE - 03

Description: Check the structure of the model.
Precondition: The flow which you want to visualize exists on the OpenML website and is originally created by one of the supported libraries or specifications (Keras, PyTorch, MXNet or ONNX). The visualization package is already installed. The OpenML library is already installed.
Requirements fulfilled: URF - 19.

Input specification:	***Output specification:***
1. Run the `visualize.py` file from the OpenML library.	
	2. The loading process finishes and outputs in the console the IP and port number of the local Dash website, possibly with a delay of a few seconds.
3. Open the local website in a browser.	
4. Find the id of the flow on the OpenML website.	
5. Input the id in the run field of the local website and load it.	
	6. The structure of the model is rendered.
7. Check the structure of the model.	

4 | Test procedure

This chapter describes the procedure that should be followed to correctly run the acceptance tests. The acceptance tests which the OpenML Support Squad describes are strictly related to the extension packages made by the team in project "Sharing deep learning models" and the visualization prototype. These include the following extensions: Keras, PyTorch, ONNX and MXNet. Tests are separated in groups according to which library they refer to. In this section the procedure is provided to run tests that are in the scope of this project.

4.1 KerasExtension test procedure

In this section the procedure is provided to run tests that are meant for testing the Keras extension. To follow the procedure the tester must pick a given model made using Keras which they are going to use throughout the testing procedure below.

4.1.1 Test procedure identifier

AT - 01

4.1.2 Purpose

This procedure executes all acceptance tests that are implemented by OpenML Support Squad for the Keras extension. These check whether the implemented extension adheres to the requirements. Only one model should be used for the complete procedure in this section. If the model used does not follow the prerequisites specified in the ATD - 01, this means that OpenML Support Squad is not liable for failing other acceptance tests, because the reasons for failing are outside of the scope of our project. If all the tests pass, the requirements specified in the URD [1] will be fulfilled.

4.1.3 Procedure steps

1. Execute ATD - 01
2. Execute ATA - 01
3. Execute ATB - 01
4. Execute ATA - 02
5. Execute ATB - 01
6. Execute ATC - 01 (***Note:*** *If steps 3 and 4 of this test procedure have been successful, the tester can continue with ATC - 01 by starting directly from step 8.*)

4.2 PytorchExtension test procedure

In this section the procedure is provided to run tests that are meant for testing the PyTorch extension. To follow the procedure the tester must pick a given model made using PyTorch which they are going to use throughout the testing procedure below.

4.2.1 Test procedure identifier

AT - 02

4.2.2 Purpose

This procedure executes all acceptance tests that are implemented by OpenML Support Squad for the PyTorch extension. These check whether the implemented extension adheres to the requirements. Only one model should be used for the complete procedure in this section. If the model used does not follow the prerequisites specified in the ATD - 02, this means that OpenML Support Squad is not liable for failing other acceptance tests, because the reasons for failing are outside of the scope of our project. If all the tests pass, the requirements specified in the URD [1] will be fulfilled.

4.2.3 Procedure steps

1. Execute ATD - 02
2. Execute ATA - 03
3. Execute ATB - 02
4. Execute ATA - 04
5. Execute ATB - 02
6. Execute ATC - 02 (***Note:*** *If steps 3 and 4 of this test procedure have been successful, the tester can continue with ATC - 02 by starting directly from step 8.*)

4.3 MXNetExtension test procedure

In this section the procedure is provided to run tests that are meant for testing the MXNet extension. To follow the procedure the tester must pick a given model made using MXNet which they are going to use throughout the testing procedure below.

4.3.1 Test procedure identifier

AT - 03

4.3.2 Purpose

This procedure executes all acceptance tests that are implemented by OpenML Support Squad for the MXNet extension. These check whether the implemented extension adheres to the requirements. Only one model should be used for the complete procedure in this section. If the model used does not follow the prerequisites specified in the ATD - 03, this means that OpenML Support Squad is not liable for failing other acceptance tests, because the reasons for failing are outside of the scope of our project. If all the tests pass, the requirements specified in the URD [1] will be fulfilled.

4.3.3 Procedure steps

1. Execute ATD - 03
2. Execute ATA - 05
3. Execute ATB - 03
4. Execute ATA - 06
5. Execute ATB - 03
6. Execute ATC - 03 (***Note:*** *If steps 3 and 4 of this test procedure have been successful, the tester can continue with ATC - 03 by starting directly from step 8.*)

4.4 ONNXExtension test procedure

In this section the procedure is provided to run tests that are meant for testing the ONNX extension. To follow the procedure the tester must pick a given model made using ONNX which they are going to use throughout the testing procedure below.

4.4.1 Test procedure identifier

AT - 04

4.4.2 Purpose

This procedure executes all acceptance tests that are implemented by OpenML Support Squad for the ONNX extension. These check whether the implemented extension adheres to the requirements. Only one model should be used for the complete procedure in this section. If the model used does not follow the prerequisites specified in the ATD - 04, this means that OpenML Support Squad is not liable for failing other acceptance tests, because the reasons for failing are outside of the scope of our project. If all the tests pass, the requirements specified in the URD [1] will be fulfilled.

4.4.3 Procedure steps

1. Execute ATD - 04
2. Execute ATA - 07

3. Execute ATB - 04

4. Execute ATC - 04 (***Note:*** *If steps 3 and 4 of this test procedure have been successful, the tester can continue with ATC - 04 by starting directly from step 8.*)

4.5 Visualization

In this section the procedure is provided to run tests that are meant for testing the visualization module of the DL Extension Library. To follow the procedure the tester must pick a given existing OpenML flow created by one of the supported deep learning libraries and specifications (Keras, PyTorch, MXNet or ONNX) and a run which contains training data. These are going to use throughout the testing procedure below.

4.5.1 Test procedure identifier

AT - 05

4.5.2 Purpose

This procedure executes all acceptance tests that are implemented by OpenML Support Squad for the visualization module of the DL Extension Library. These check whether the visualization adheres to the requirements. If all the tests pass, the requirements specified in the URD [1] will be fulfilled.

4.5.3 Procedure steps

1. Execute ATE - 01

2. Execute ATE - 02

3. Execute ATE - 03

5 | Test reports

This chapter contains a test report for each extension implemented as well as the visualization. Each test report shows whether all the test procedures associated to the extension/visualization have passed or not. It also shows the date when they passed, who was the tester, the witnesses and the requirements fulfilled.

KerasExtension ATR - 01

Procedure tested: ATD - 01
Requirements fulfilled: URC - 01, URC - 09, URC - 34
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 01
Requirements fulfilled: URF - 03; URF - 17; URF - 20; URC - 37
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 01
Requirements fulfilled: URF - 04; URC - 40
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 02
Requirements fulfilled: URF - 03; URF - 17; URF - 21; URC - 37
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 01
Requirements fulfilled: URF - 04; URC - 40
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATC - 01
Requirements fulfilled: URC - 18; URC - 20
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

PyTorchExtension ATR - 02

Procedure tested: ATD - 02
Requirements fulfilled: URC - 02, URC - 10, URC - 34
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 03
Requirements fulfilled: URF - 05; URF - 17; URF - 22; URC - 38
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 02
Requirements fulfilled: URF - 06; URC - 41
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 04
Requirements fulfilled: URF - 05; URF - 17; URF - 23; URC - 38
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 02
Requirements fulfilled: URF - 06; URC - 41
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATC - 02
Requirements fulfilled: URC - 19; URC - 21
Date: 21/06/2019

Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

MXNetExtension ATR - 03

Procedure tested: ATD - 03
Requirements fulfilled: URC - 06, URC - 14, URC - 34
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 05
Requirements fulfilled: URF - 09; URF - 17; URF - 24; URC - 43
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 03
Requirements fulfilled: URF - 10; URC - 45
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 06
Requirements fulfilled: URF - 09; URF - 17; URF - 25; URC - 43
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 03
Requirements fulfilled: URF - 10; URC - 45
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATC - 03
Requirements fulfilled: URC - 27; URC - 29
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

ONNXExtension ATR - 04

Procedure tested: ATD - 04
Requirements fulfilled: URC - 03, URC - 11, URC - 34
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATA - 07
Requirements fulfilled: URF - 01; URF - 17; URC - 39
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATB - 04
Requirements fulfilled: URF - 02; URC - 42
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATC - 04
Requirements fulfilled: URC - 22; URC - 23
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Visualization ATR - 05

Procedure tested: ATE - 01
Requirements fulfilled: URF - 18
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATE - 02
Requirements fulfilled: URF - 18
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Procedure tested: ATE - 03
Requirements fulfilled: URF - 19
Date: 21/06/2019
Tester: Joaquin Vanschoren
Witness: Ion Barosan & OpenML Support Squad & The Project Managers
Verdict: Pass

Bibliography

[1] OpenML Support Squad. *DL Extension Library, URD User Requirement Document version 1.0.0.*

[2] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[3] OpenML. *OpenML Documentation.* URL: `https://docs.openml.org/` (visited on 04/26/2019).

[4] Wikipedia. *Apache MXNet.* URL: `https://en.wikipedia.org/wiki/Apache_MXNet` (visited on 04/26/2019).

[5] Keras. *Keras homepage.* URL: `https://keras.io/` (visited on 04/26/2019).

[6] Wikipedia. *Artificial Neural Network.* URL: `https://en.wikipedia.org/wiki/Artificial_neural_network` (visited on 04/30/2019).

[7] PyTorch. *PyTorch homepage.* URL: `https://pytorch.org/` (visited on 09/05/2019).