3rd July 2019

User Requirements Document

Version 2.0.0

OpenML

Members
Andrei Danila | 1034559
Bogdan Enache | 1035066
Gergana Goncheva | 1037010
Loïc Alexander Hijl | 1002745
Adrian-Stefan Mares | 0993873
Veselin Minev | 1014541
Thanh-Dat Mathieu 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

Supervisors
Erik Luit
Ion Barosan

Customer
Joaquin Vanschoren

Abstract

OpenML is a platform for sharing machine learning models, datasets, tasks and benchmarks. This document contains the user requirements for the Deep Learning (DL) Extension Library for OpenML, in which support for sharing deep learning models was implemented by team OpenML Support Squad, as well as for the visualization module related to this project. This document complies with the European Space Agency (ESA) standards [1]. All requirements have been discussed with the client, Joaquin Vanschoren, and have been agreed upon.

Contents

Document Status Sheet

Document Title: User Requirements Document
Identification: URD/2.0.0
Version: 2.0.0
Authors: A.Danila, B.Enache, G.Goncheva, L.A.Hijl, A.Mares, V.Minev, T.M.Nguyen, A.L.L.Nijhuis, C.T. 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
0.1.0	26-04-2019	All authors	First draft requirements
0.1.1	30-04-2019	All authors	First draft finished
0.1.2	01-05-2019	All authors	Made changes based on feedback
0.1.3	02-05-2019	All authors	Fixes after all feedback
0.2.0	02-05-2019	All authors	Second draft finished
0.2.1	02-05-2019	All authors	Fixed grammatical errors after supervisor feedback
1.0.0	02-05-2019	All authors	Fixed after client feedback
1.0.1	06-06-2019	All authors	Definition of OpenML flow from the glossary changed. The URCs about size and number of layers were modified to be more specific. The URFs from Web Visualization have been rephrased.
1.1.0	14-06-2019	All authors	Implement feedback on the URD, such as clarifying and defining things more clearly
1.2.0	18-06-2019	S. Zarkova	Proofreading and corrections
2.0.0	24-06-2019	L.A. Hijl	Final proofreading and corrections

Document Change Records

Date	Section	Author(s)	Reason
06-06-2019	Glossary & 3.2 & 3.1	All authors	Clarified the definition of OpenML. Specified the size and number of layers in some URCs. Rephrasing the URFs for web visualization.
14-06-2019	Whole document	All authors	Implemented feedback for clarifying various sections.

1 | Introduction

1.1 Purpose

This User Requirements Document (URD) contains the requirements for the project "Sharing deep learning models", an extension for the OpenML platform. These requirements are negotiated and agreed upon between the client, Joaquin Vanschoren, and the OpenML Support Squad. The listed requirements in this document will be implemented in the end products - the DL Extension Library and the visualization module - according to their respective priorities. The intended readership audience consists of researchers and machine learning specialists / enthusiasts. Any further changes to this document and the requirements within require the full consent of both parties, i.e. OpenML Support Squad and Joaquin Vanschoren.

The purpose of project "Sharing deep learning models" is to create an extension for the already existent OpenML Python Application Programming Interface (API) and provide a visualization module. The extension, called DL Extension Library, represents the biggest part of the product that OpenML Support Squad has to deliver. DL Extension Library will enable sharing deep learning models to the OpenML platform. The visualization module allows visualization of a model's structure and performance metrics in Dash.

1.2 Scope

OpenML is an open-source online machine learning platform for sharing and organizing data, machine learning algorithms and experiments. It allows people all over the world to collaborate and build on each other's latest findings, data or results. OpenML is designed to create a seamless network that can be integrated into existing environments, regardless of the tools and infrastructure used. It allows users to upload datasets, models (called OpenML flows) and tasks for OpenML flows. It further provides support for uploading runs that combine a model with a task for it, to allow users to compare results.

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 will not modify the OpenML Python API since it is outside of the scope of the project "Sharing deep learning models".

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

The DL Extension Library is built to provide support for the following deep learning libraries and specifications on the OpenML platform: Keras, PyTorch, Apache MXNet (MXNet) and Open

Neural Network eXchange (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.

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

An overview of the environment for this project can be seen in figure 1.1.

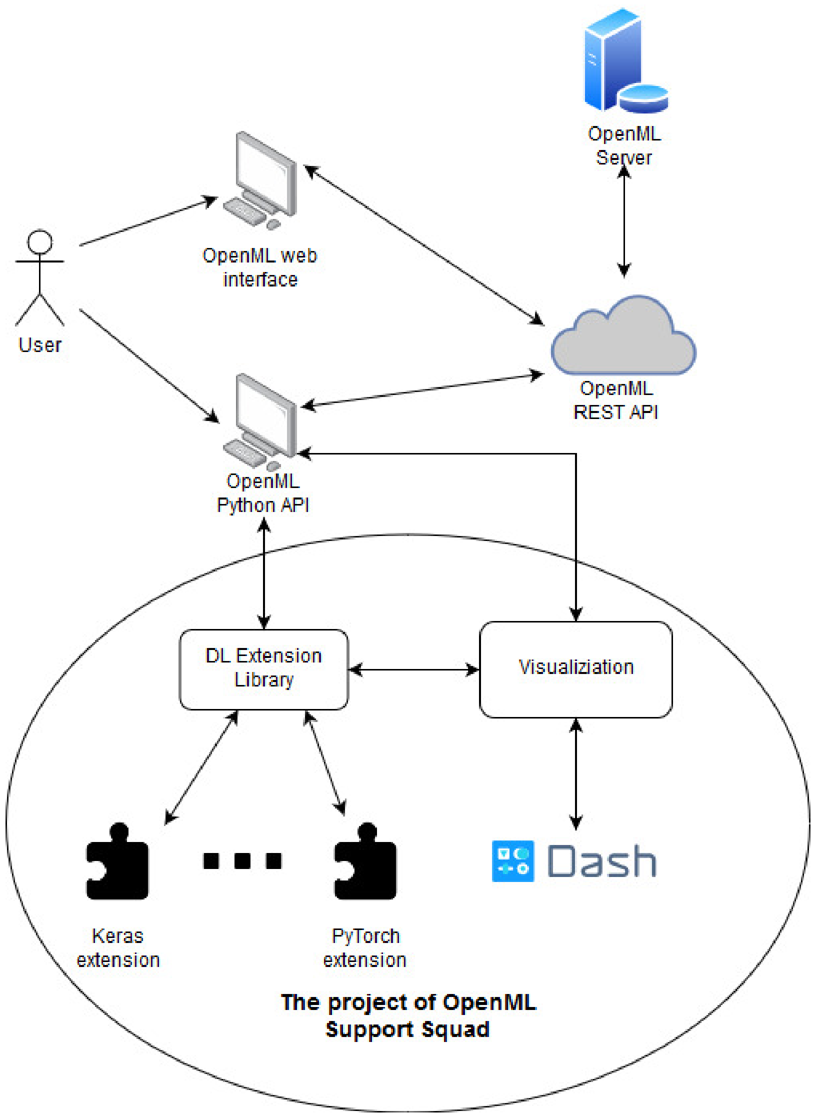

Figure 1.1: Overview diagram of the environment

1.3 Definitions and Abbreviations

1.3.1 Definitions

DL Extension Library	The product to be created in order to convert the deep learning models and the model transfer specifications (i.e. ONNX and MLflow) into a format supported by the OpenML platform.
Keras*	A placeholder for Keras and PyTorch.
OpenML Main REST API	An interface supporting the interaction between the OpenML Python API and the OpenML platform. "Has all main functions to download OpenML data or share your own" [2].
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" [2].
OpenML flow	The representation of untrained 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)" [3].
Caffe2	"CAFFE2 (Convolutional Architecture for Fast Feature Embedding) is a open-source deep learning framework" [4]. "Caffe2 comes with Python and C++ APIs" [5].
Classification task	"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" [6].
Correct Conversion	The conversion is correct when converting from a model to a flow and back to a model is done without loss of information for the model.
Dash	"Dash is a user interface library for creating analytical web applications" [7].
Dataset	"Datasets are pretty straight-forward. They simply consist of a number of rows, also called instances, usually in tabular form" [2].
Deep learning model	A neural network with more than three layers. Deep learning is a subset of machine learning.
Fast.ai	"The fast.ai library simplifies training fast and accurate neural networks using modern best practices." [8].

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" [9].
Keras**	A placeholder for all implemented libraries. Can represent frameworks and specifications. Can be replaced with specific instances of Keras, PyTorch, ONNX and MXNet. Used for abstraction.
Machine learning model	A model that performs a "task without explicit instructions, relying on patterns and inference" [10] through learning from data.
Microsoft Cognitive Toolkit	"CNTK is a unified deep learning toolkit that describes neural networks as a series of computational steps via a directed graph" [11].
MLFlow specification	"An MLflow Model is a standard format for packaging machine learning models that can be used in a variety of downstream tools—for example, real-time serving through a REST API or batch inference on Apache Spark. " [12].
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" [13].
Non-redundant dependencies	The dependencies should be library-specific (based on the type of model used), only necessary dependencies being required for the proper operation of a certain type of a deep learning model.
ONNX specification	"An open format to represent represent deep learning models" [14]. The representation does not change between different deep learning frameworks.
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" [15].
Regression task	A statistical process that "attempts to determine the strength of the relationship between one dependent variable (usually denoted by Y) and a series of other changing variables (known as independent variables)" [16].

Run	"A run is a particular flow, that is algorithm, with a particular parameter setting, applied to a particular task" [2].
Sklearn	A Python library that provides machine learning algorithms.
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" [2].

1.3.2 Abbreviations

API	Application Programming Interface.
C2	Caffe2.
CNTK	Microsoft Cognitive Toolkit.
DL	Deep Learning.
ESA	European Space Agency.
FA	fast.ai.
KR	Keras.
MXNet	Apache MXNet.
ONNX	Open Neural Network eXchange.
PT	PyTorch.
URC	User Requirements Constraint.
URD	User Requirements Document.
URF	User Requirements Functional.

1.4 List of references

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

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

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

[4] Wikipedia. *Caffe (software*. URL: `https://en.wikipedia.org/wiki/Caffe_(software)` (visited on 26/04/2019).

[5] Caffe2. *Caffe2 homepage*. URL: `https://caffe2.ai/` (visited on 26/04/2019).

[6] Wikipedia. *Statistical classification*. URL: `https://en.wikipedia.org/wiki/Statistical_classification` (visited on 01/05/2019).

[7] Kyle Kelley. *Introducing Dash*. URL: `https://medium.com/@plotlygraphs/introducing-dash-5ecf7191b503` (visited on 05/06/2019).

[8] fast.ai. *The fastai deep learning library, plus lessons and tutorials*. URL: `https://github.com/fastai/fastai` (visited on 30/04/2019).

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

[10] Wikipedia. *Machine Learning*. URL: `https://en.wikipedia.org/wiki/Machine_learning` (visited on 26/04/2019).

[11] Microsoft. *Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit (github)*. URL: `https://github.com/Microsoft/CNTK` (visited on 26/04/2019).

[12] MLflow. *MLflow Models*. URL: `https://mlflow.org/docs/latest/models.html` (visited on 26/04/2019).

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

[14] ONNX. *ONNX homepage*. URL: `https://onnx.ai/` (visited on 26/04/2019).

[15] PyTorch. *PyTorch homepage*. URL: `https://pytorch.org/` (visited on 26/04/2019).

[16] Brian Beers. *Regression Definition*. URL: `https://www.investopedia.com/terms/r/regression.asp` (visited on 01/05/2019).

[17] Agile Business Consortium. *MoSCoW Prioritisation*. URL: `https://www.agilebusiness.org/content/moscow-prioritisation-0` (visited on 26/04/2019).

[18] Wikipedia. *MoSCoW method*. URL: `https://en.wikipedia.org/wiki/MoSCoW_method` (visited on 26/04/2019).

1.5 Overview

The next chapter provides a general description of the DL Extension Library and the visualization module that OpenML Support Squad will work on. Section 2.1 is about the product perspective. In Section 2.2 and 2.3 the capability and constraint requirements are listed. Consequently, in Section 2.4 and 2.5, the user characteristics and a description of the environment are given. In Section 2.6 the assumptions and dependencies of the DL Extension Library and the visualization module are explained. Next is chapter 3 where the requirements and their respective priorities are listed and divided into two categories. These categories are further split into subcategories. Section 3.1 discusses all capability requirements and section 3.2 discusses the constraint requirements. Finally, in the appendix, five use cases and their corresponding diagrams are provided and explained.

2 | General Description

This section describes the purpose of the DL Extension Library and the visualization module. It also describes the functionality that they provide. Moreover, the constraints in terms of supported machine learning models and libraries are listed. A short description of the environment is provided, which explains how the OpenML platform is set up. Finally, OpenML Support Squad, in agreement with the client, Joaquin Vanschoren, decided upon some assumptions and dependencies that need to be taken into consideration when extending the OpenML platform to support deep learning models.

2.1 Product Perspective

OpenML has been created to give researchers and data scientists an online platform on which they can share models built on different frameworks. Furthermore, the platform supports the creation of tasks, hence allowing different machine learning models to be compared and evaluated on their performance for the same task. This functionality is very useful since users can pick up the best models and refine them further.

The problem with the OpenML Python API currently is that it supports only the Sklearn framework, which represents a small part of all machine learning frameworks. Sklearn does also not have deep learning functionality, which is quite useful in machine learning. This is why the DL Extension Library is needed.

For clarification, OpenML is not a platform which runs and trains machine learning models, it only deals with sharing and comparing them. All models can be retrieved from the platform by the user, who can, later on, run and train them using their local machine. However, OpenML does provide users with data sets so that the machine learning models can be trained.

OpenML supports the two main types of machine learning tasks: Supervised Regression and Supervised Classification. These tasks can be performed by all types of machine learning models, including the deep learning ones, which OpenML does not support yet. Since deep learning models are important for the Machine learning community, OpenML needs to be extended in order to support them.

2.2 General Capabilities

The DL Extension Library and visualization module will have the capabilities described below.

2.2.1 Library Conversion

The already existing OpenML platform enables its users to share Machine Learning datasets, tasks, pipelines and benchmarks. However, it does not support the sharing of deep learning models by default. As this is a capability which is of interest to the users, the DL Extension Library will provide the necessary conversion functionality, enabling users to retrieve and publish deep learning models on the OpenML platform. Furthermore, this will allow the users to train deep learning models, make predictions on them and share the resulting OpenML runs. For OpenML to support deep learning models, a conversion has to be performed from a model of an existing deep learning library to an OpenML flow and vice versa. OpenML flow is the OpenML supported format. DL Extension Library will supports several popular deep learning libraries, including Keras and PyTorch.

2.2.2 Web Visualization

All deep learning models supported by the DL Extension Library on the OpenML platform can be represented in the ONNX format. For regular machine learning models, OpenML automatically analyzes the model data, visualizes results and computes data characteristics. By successfully converting deep learning models into OpenML format, the functionality described above automatically applies to deep learning models as well. However, for deep learning models the user might deem it interesting to have a node-based graph representation of the neural network itself. The visualization module implements this by using the ONNX representation of a deep learning model. Additionally, it provides the user with a visual representation of the learning curve for deep learning models. Since the visualization of a deep learning model depends on successfully uploading the deep learning model to OpenML. This is implemented only after Deep learning models can be converted and shared successfully.

2.2.3 Task Types

When working with deep learning models, users are generally interested in a specific task type, which is one of the following: classification task or regression task. OpenML itself does not provide support for any type of deep learning task, so the DL Extension Library serves as an extension to the existing OpenML Python API. The DL Extension Library will allow users to run deep learning models on classification and regression tasks, using the OpenML platform.

2.3 General Constraints

The DL Extension Library and visualization module conform to the constraints described below.

2.3.1 Visualization

The visualization module utilizes the Dash user interface library to provide both a visual presentation of the deep learning neural networks and the learning curve. Dash is a Python framework for building analytical applications, and the visualization module must be built using it. This is how the customer wants it.

2.3.2 Data Reliability

The DL Extension Library correctly converts any model from the supported frameworks into an OpenML flow format. Furthermore, the user should be able to retrieve the model from the OpenML platform. Therefore, the OpenML flow should be able to correctly convert back to the initial model. In the given context, correct conversion implies that the structure of the deep learning model is persistent.

2.3.3 Environment

The environment in which the DL Extension Library will operate relies solely on the OpenML Python API. The DL Extension Library extends this API. All necessary operations that can be performed on deep learning models such as uploading, deleting and analyzing are done through OpenML Python API. In order to function consistently within this environment, we ensure that the DL Extension Library is compatible with Python version 3.5. Additionally, there is the aim at making it compatible with versions 3.6 and 3.7 as well, following the client's requirements.

2.3.4 Dependencies

To provide the users with the desired functionality without having to download all supported deep learning libraries (i.e. Keras, PyTorch etc.), the support for each library is implemented in a separate package. Consequently, users are required to install only the package that provides support for their desired deep learning library. For example, if the user is interested in working with a Keras model, he/she would need to install only the package that provides conversion between Keras models and OpenML flows. PyTorch and other deep learning libraries will not be required for this particular instance.

As deep learning models can easily become very complex, it is important that there are constraints on the maximum size of the models that the DL Extension Library supports. In the implementation for OpenML, there are existing constrains on the size of the models which can be published to the platform. Therefore, the DL Extension Library is limited to deep learning models of up to 1GB and up to 30 layers.

2.4 User Characteristics

The OpenML Python API makes a distinction between registered users (i.e. those who have an API key) and unregistered users(i.e. those who do not have an API key). It then decides what functions are allowed for each user based on this distinction. Thus, it is important to clarify whether the DL Extension Library will make a similar distinction or not. Since the DL Extension Library deals only with converting deep learning models into OpenML flows and vice versa, there will be no need to authenticate in order to perform it. Making a distinction between registered and unregistered users is not the purpose of the DL Extension Library because all interactions with the platform are made through the OpenML Python API and not through the DL Extension Library.

2.5 Environment description

A diagram illustrating the environment is shown in figure 2.1. The main component is the DL Extension Library, which is responsible for converting a given deep learning model to an OpenML flow. The given deep learning model can be either provided directly (i.e. a model built on Keras, PyTorch or any other supported library) or as a specification (i.e. ONNX or MLFlow specification).

The DL Extension Library is built on the OpenML Extension API, which is already provided and is part of the OpenML Python API. The OpenML Python API is responsible for any interactions with the OpenML REST API.

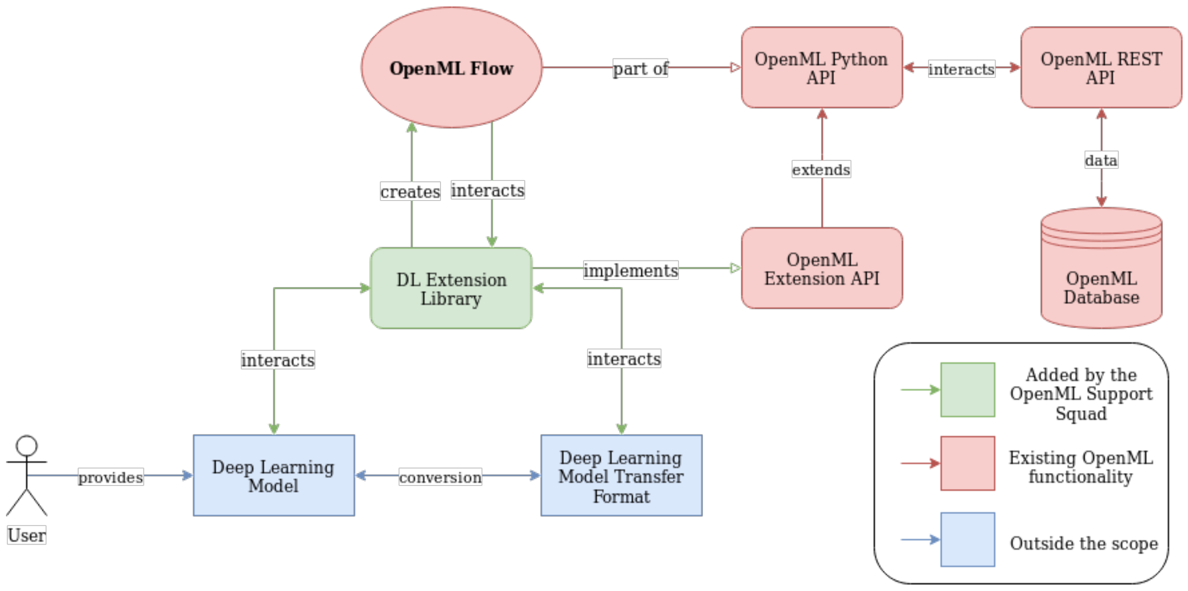

Figure 2.1: Environment diagram

2.6 Assumptions and Dependencies

In order for the DL Extension Library to function correctly the assumptions and dependencies presented below must be satisfied.

- OpenML Main REST API does not change for the duration of the project.
- OpenML Python API does not change for the duration of the project.
- OpenML Extension API does not change for the duration of the project.
- OpenML flow model representation does not change for the duration of the project.
- Conversion of a Keras model to an ONNX specification and vice versa is provided by third parties.
- Conversion of a PyTorch model to an ONNX specification and vice versa is provided by third parties.
- Conversion of an Apache MXNet model to an ONNX specification and vice versa is provided by third parties.
- Conversion of a Caffe2 model to an ONNX specification and vice versa is provided by third parties.
- Conversion of a fast.ai model to an ONNX specification and vice versa is provided by third parties.

- Conversion of a Microsoft Cognitive Toolkit model to an ONNX specification and vice versa is provided by third parties.

3 | Specific Requirements

In this chapter a list of requirements for the project "Sharing deep learning models" is provided. These requirements are divided into two sections: *capability* and *constraint* requirements. These are further split into subsections. The *capability* requirements are denoted by User Requirements Functional (URF) and are divided into three categories: *converting library*, *web visualization* and *task types*. These requirements specify what the user can do with the DL Extension Library. The *constraint* requirements are denoted by User Requirements Constraint (URC) and are divided into four categories: *general*, *data reliability*, *environment* and *performance*. These requirements denote the limits that the system cannot cross while operating. The MoSCoW method [17] is used to prioritize requirements with the help of four categories. These categories are the following:

- **Must have**: requirements that must be implemented for the product to function properly. These determine whether the product is a Minimum Usable Subset. [18].
- **Should have**: requirements that are important for the project but are not seen as necessary to be implemented for the project to be successfully completed. Note that *Must have* requirements could be downgraded to *Should have* requirements, but only if agreed by all stakeholders.
- **Could have**: requirements that are seen as optional and can enhance the user experience. These requirements can be implemented if time allows and the more important requirements are already implemented.
- **Won't have**: requirements that will not be implemented by the development team in the current project and hence are not part of the scope of this project. They may be picked up by a new development team.

In total, this chapter contains **31** functional requirements and **48** constraint requirements.

3.1 Capability Requirements

3.1.1 Format conversion

URF - 01 The DL Extension Library shall convert an ONNX specification to an OpenML flow.	Must have
URF - 02 The DL Extension Library shall convert an OpenML flow to an ONNX specification.	Must have
URF - 03 The DL Extension Library shall convert a Keras (KR) model to an OpenML flow.	Must have

URF - 04 The DL Extension Library shall convert an OpenML flow to a KR model.	Must have
URF - 05 The DL Extension Library shall convert a PyTorch (PT) model to an OpenML flow.	Must have
URF - 06 The DL Extension Library shall convert an OpenML flow to a PT model.	Must have
URF - 07 The DL Extension Library shall convert a MLFlow specification to an OpenML flow.	Should have
URF - 08 The DL Extension Library shall convert an OpenML flow to a MLFlow specification.	Should have
URF - 09 The DL Extension Library shall convert a MXNet model to an OpenML flow.	Should have
URF - 10 The DL Extension Library shall convert an OpenML flow to a MXNet model.	Should have
URF - 11 The DL Extension Library shall convert a Caffe2 (C2) model to an OpenML flow.	Should have
URF - 12 The DL Extension Library shall convert an OpenML flow to a C2 model.	Should have
URF - 13 The DL Extension Library shall convert a fast.ai (FA) model to an OpenML flow.	Could have
URF - 14 The DL Extension Library shall convert an OpenML flow to a FA model.	Could have
URF - 15 The DL Extension Library shall convert a Microsoft Cognitive Toolkit (CNTK) model to an OpenML flow.	Could have
URF - 16 The DL Extension Library shall convert an OpenML flow to a CNTK model.	Could have

3.1.2 Web Visualization

URF - 17	Must have
For a deep learning model uploaded on the OpenML website, a hyperparameter representation of the layers shall be created in the OpenML flow.	
URF - 18	Should have
For a deep learning model uploaded on the OpenML website, the learning curve graph of the models shall be visualized.	
URF - 19	Should have
For a deep learning model uploaded on the OpenML website, its neural network structure shall be visualized.	

3.1.3 Task Types

URF - 20	Must have
The DL Extension Library shall accept KR classification tasks.	
URF - 21	Must have
The DL Extension Library shall accept KR regression tasks.	
URF - 22	Must have
The DL Extension Library shall accept PT classification tasks.	
URF - 23	Must have
The DL Extension Library shall accept PT regression tasks.	
URF - 24	Should have
The DL Extension Library shall accept MXNet classification tasks.	
URF - 25	Should have
The DL Extension Library shall accept MXNet regression tasks.	
URF - 26	Should have
The DL Extension Library shall accept C2 classification tasks.	
URF - 27	Should have
The DL Extension Library shall accept C2 regression tasks.	
URF - 28	Could have
The DL Extension Library shall accept FA classification tasks.	
URF - 29	Could have
The DL Extension Library shall accept FA regression tasks.	
URF - 30	Could have
The DL Extension Library shall accept CNTK classification tasks.	
URF - 31	Could have
The DL Extension Library shall accept CNTK regression tasks.	

3.2 Constraint requirements

3.2.1 General

URC - 01 The DL Extension Library shall correctly convert Keras models of up to 1 GB in size.	Must have
URC - 02 The DL Extension Library shall correctly convert PyTorch models of up to 1 GB in size.	Must have
URC - 03 The DL Extension Library shall correctly convert ONNX specifications of up to 1 GB in size.	Must have
URC - 04 The DL Extension Library shall correctly convert C2 models of up to 1 GB in size.	Should have
URC - 05 The DL Extension Library shall correctly convert MLFlow specification specifications of up to 1 GB in size.	Should have
URC - 06 The DL Extension Library shall correctly convert MXNet models of up to 1 GB in size.	Should have
URC - 07 The DL Extension Library shall correctly convert FA models of up to 1 GB in size.	Could have
URC - 08 The DL Extension Library shall correctly convert CNTK models of up to 1 GB in size.	Could have
URC - 09 The DL Extension Library shall correctly convert Keras models of up to 30 layers.	Must have
URC - 10 The DL Extension Library shall correctly convert PyTorch models of up to 30 layers.	Must have
URC - 11 The DL Extension Library shall correctly convert ONNX specification of up to 30 layers.	Must have
URC - 12 The DL Extension Library shall correctly convert C2 models of up to 30 layers.	Should have

URC - 13 The DL Extension Library shall correctly convert MLFlow specification models of up to 30 layers.	Should have
URC - 14 The DL Extension Library shall correctly convert MXNet models of up to 30 layers.	Should have
URC - 15 The DL Extension Library shall correctly convert FA models of up to 30 layers.	Could have
URC - 16 The DL Extension Library shall correctly convert CNTK models of up to 30 layers.	Could have
URC - 17 Visualization shall use Dash.	Should have

3.2.2 Data Reliabilty

URC - 18 The DL Extension Library shall correctly convert a KR model to an OpenML flow.	Must have
URC - 19 The DL Extension Library shall correctly convert a PT model to an OpenML flow.	Must have
URC - 20 The DL Extension Library shall correctly convert an OpenML flow to a KR model.	Must have
URC - 21 The DL Extension Library shall correctly convert an OpenML flow to a PT model.	Must have
URC - 22 The DL Extension Library shall correctly convert an ONNX specification to an OpenML flow.	Must have
URC - 23 The DL Extension Library shall correctly convert an OpenML flow to an ONNX specification.	Must have
URC - 24 The DL Extension Library shall correctly convert a C2 model to an OpenML flow.	Should have
URC - 25 The DL Extension Library shall correctly convert an OpenML flow to a MLFlow specification.	Should have

URC - 26 The DL Extension Library shall correctly convert a MLFlow specification to an OpenML flow.	Should have
URC - 27 The DL Extension Library shall correctly convert an OpenML flow to a MXNet model.	Should have
URC - 28 The DL Extension Library shall correctly convert an OpenML flow to a C2 model.	Should have
URC - 29 The DL Extension Library shall correctly convert a MXNet model to an OpenML flow.	Should have
URC - 30 The DL Extension Library shall correctly convert a FA model to an OpenML flow.	Could have
URC - 31 The DL Extension Library shall correctly convert a CNTK model to an OpenML flow.	Could have
URC - 32 The DL Extension Library shall correctly convert an OpenML flow to a FA model.	Could have
URC - 33 The DL Extension Library shall correctly convert an OpenML flow to a CNTK model.	Could have

3.2.3 Environment

URC - 34 The DL Extension Library shall be compatible with Python 3.5.	Must have
URC - 35 The DL Extension Library shall be compatible with Python 3.6.	Should have
URC - 36 The DL Extension Library shall be compatible with Python 3.7.	Should have

3.2.4 Dependencies

URC - 37	The DL Extension Library shall convert a KR model to an OpenML flow using only non-redundant dependencies.	Must have
URC - 38	The DL Extension Library shall convert a PT model to an OpenML flow using only non-redundant dependencies.	Must have
URC - 39	The DL Extension Library shall convert an ONNX specification to an OpenML flow using only non-redundant dependencies.	Must have
URC - 40	The DL Extension Library shall convert an OpenML flow to a KR model using only non-redundant dependencies.	Must have
URC - 41	The DL Extension Library shall convert an OpenML flow to a PT model using only non-redundant dependencies.	Must have
URC - 42	The DL Extension Library shall convert an OpenML flow to an ONNX specification using only non-redundant dependencies.	Must have
URC - 43	The DL Extension Library shall convert a MXNet model to an OpenML flow using only non-redundant dependencies.	Should have
URC - 44	The DL Extension Library shall convert a C2 model to an OpenML flow using only non-redundant dependencies.	Should have
URC - 45	The DL Extension Library shall convert an OpenML flow to a MXNet model using only non-redundant dependencies.	Should have
URC - 46	The DL Extension Library shall convert an OpenML flow to a C2 model using only non-redundant dependencies.	Should have
URC - 47	The DL Extension Library shall convert an OpenML flow to a CNTK model using only non-redundant dependencies.	Could have
URC - 48	The DL Extension Library shall convert a CNTK model to an OpenML flow using only non-redundant dependencies.	Could have

A | Use cases

A general, abstracted overview of a use case diagram is depicted on figure A.1.

UC 1 Run a Keras** model

Goal: Publish on the OpenML website the run of the Keras** model together with the OpenML flow representation of the model, depicted on figure A.2.
Summary: The user runs and publishes the model.
Preconditions: The user has a Keras** model and a classification/regression task.
Postconditions: The run together with the OpenML flow representation of the model are uploaded on the website.
Covered requirements: URF - 01, URF - 03, URF - 05, URF - 07, URF - 09, URF - 11, URF - 13, URF - 15, URF - 17 and from URF - 20 to URF - 31
Steps:

Actor actions	*Response*
1. *User runs the Keras** model on the task using OpenML Python API.*	
	2. The OpenML Python API runs the Keras** model using the DL Extension Library.
	3. *The OpenML Python API stores the results in the OpenML run.*
	4. The DL Extension Library converts the Keras** model into an OpenML flow.
	5. DL Extension Library stores the hyperparameter representation of the model into the OpenML flow.
	6. *The OpenML Python API stores the OpenML flow in the OpenML run.*
	7. *The OpenML Python API returns the OpenML run.*
8. *User publishes the OpenML run to OpenML using the OpenML Python API.*	
	9. *The model is published on the OpenML website.*
	10. *The hyperparameter representation of the model is displayed.*

UC 2 Download an OpenML flow from the OpenML website

Goal: Download an OpenML flow and convert to a Keras** model, depicted on figure A.3.
Summary: One user downloads an OpenML flow and converts to a Keras** model.
Preconditions: The OpenML flow exists on the OpenML website.
Postconditions: The OpenML flow is correctly converted to a Keras** model.
Covered requirements: URF - 02, URF - 04, URF - 06, URF - 08, URF - 10, URF - 12, URF - 14 and URF - 16
Steps:

Actor actions	*Response*
1. *User inserts the id of the OpenML flow and the run he wants to visualize into Dash.*	
	2. The visualization package retrieves the OpenML flow and the run from the OpenML website using OpenML Python API.
	3. The visualization package passes the data to Dash.
	4. *Dash plots the structure and the learning curve graph.*

UC 3 Run and download a Keras** model

Goal: Retrieve a published Keras** model, depicted on figure A.4.
Summary: The user runs and publishes a Keras* model and the same model is retrieved afterwards by another user.
Preconditions: User 1 has a Keras** model and a classification/regression task.
Postconditions: The Keras** model of user 1 is the same as the retrieved Keras** model of user 2.
Covered requirements: From URF - 01 to URF - 17 and from URF - 20 to URF - 31
Steps:

Actor actions	*Response*
1. *User 1 runs the Keras** model on the task using OpenML Python API.*	
	2. The OpenML Python API runs the Keras** model using the DL Extension Library.

	3. *The OpenML Python API stores the results in the OpenML run.*
	4. The DL Extension Library converts the Keras** model into an OpenML flow.
	5. DL Extension Library stores the hyperparameter representation of the model into the OpenML flow.
	6. *The OpenML Python API stores the OpenML flow in the OpenML run.*
	7. *The OpenML Python API returns the OpenML run to user 1.*
8. *User 1 publishes the OpenML run to OpenML using the OpenML Python API.*	
	9. *The model is published on the OpenML website.*
10. *User 2 downloads the same OpenML flow using the OpenML Python API.*	
	11. *The OpenML Python API retrieves the OpenML flow from the OpenML website.*
	12. The DL Extension Library converts the OpenML flow into the Keras** model.
	13. *The OpenML Python API returns the model to user 2. The model is the same as the one of user 1.*

UC 4 Visualization

Goal: Visualize the structure of a Keras** model and its learning curve graph, depicted on figure A.5.
Summary: The user utilizes the Dash local website to view the information.
Preconditions: The OpenML flow and the run of the model exist on the OpenML website. Training data is present in the run and the model corresponding to the flow is from Keras**.
Postconditions: The the structure of a Keras** model and its learning curve graph are displayed.
Covered requirements: URF - 18, URF - 19
Steps:

Actor actions	*Response*
1. The user inserts the id of the OpenML flow / run of the model in the local Dash website.	

2. The visualization package retrieves the OpenML flow / run using the OpenML Python API.

3. The visualization package sends all the data to Dash.

4. *Dash displays the structure of a Keras** model / the run's learning curve graph.*

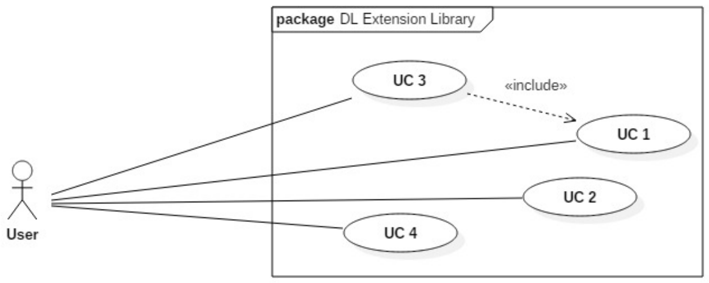

Figure A.1: General Use Case Diagram

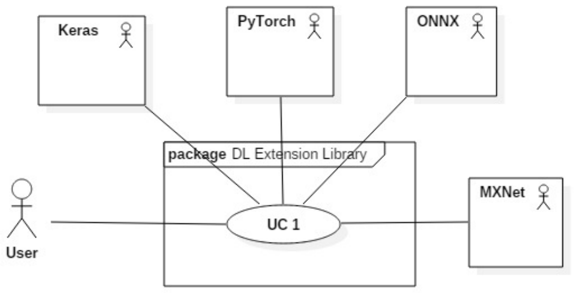

Figure A.2: Use Case Diagram - UC 1

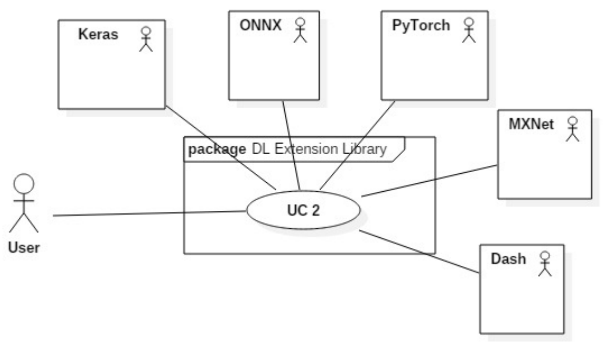

Figure A.3: Use Case Diagram - UC 2

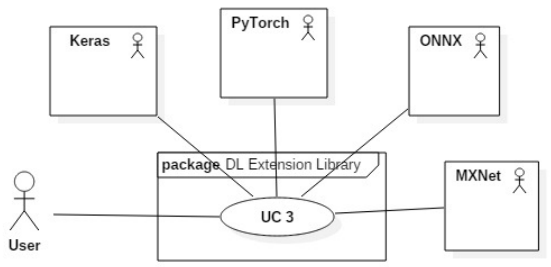

Figure A.4: Use Case Diagram - UC 3

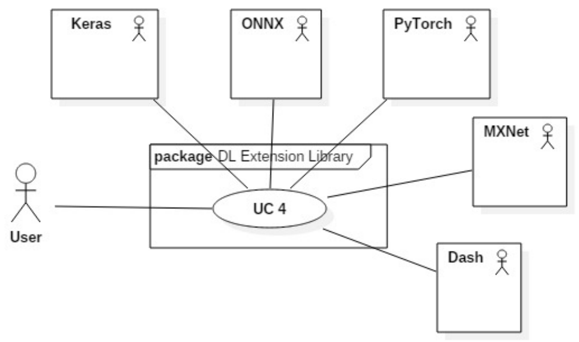

Figure A.5: Use Case Diagram - UC 4