The DeepHealth Toolkit: A Key European Free and Open-Source Software for Deep Learning and Computer Vision Ready to Exploit Heterogeneous HPC and Cloud Architectures

Marco Aldinucci, David Atienza, Federico Bolelli, Mónica Caballero, Iacopo Colonnelli, José Flich, Jon A. Gómez, David González, Costantino Grana, Marco Grangetto, Simone Leo, Pedro López, Dana Oniga, Roberto Paredes, Luca Pireddu, Eduardo Quiñones, Tatiana Silva, Enzo Tartaglione, and Marina Zapater

Abstract At the present time, we are immersed in the convergence between Big Data, High-Performance Computing and Artificial Intelligence. Technological progress in these three areas has accelerated in recent years, forcing different players like software companies and stakeholders to move quickly. The European Union is dedicating a lot of resources to maintain its relevant position in this

M. Aldinucci · I. Colonnelli · M. Grangetto · E. Tartaglione
Università degli studi di Torino, Torino, Italy

D. Atienza · M. Zapater
Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

F. Bolelli · C. Grana
Università degli studi di Modena e Reggio Emilia, Modena, Italy

M. Caballero
NTT Data Spain, Barcelona, Spain

J. Flich · J. A. Gómez (✉) · P. López · R. Paredes
Universitat Politècnica de València, Valencia, Spain
e-mail: jon@prhlt.upv.es

D. González · T. Silva
TREE Technology, Asturias, Spain

S. Leo · L. Pireddu
Center for Advanced Studies, Research and Development in Sardinia, Sardinia, Italy

D. Oniga
Softwae Imagination and Vision, București, Romania

E. Quiñones
Barcelona Supercomputing Center, Barcelona, Spain

E. Curry et al. (eds.), *Technologies and Applications for Big Data Value*,
https://doi.org/10.1007/978-3-030-78307-5_9

scenario, funding projects to implement large-scale pilot testbeds that combine the latest advances in Artificial Intelligence, High-Performance Computing, Cloud and Big Data technologies. The DeepHealth project is an example focused on the health sector whose main outcome is the DeepHealth toolkit, a European unified framework that offers deep learning and computer vision capabilities, completely adapted to exploit underlying heterogeneous High-Performance Computing, Big Data and cloud architectures, and ready to be integrated into any software platform to facilitate the development and deployment of new applications for specific problems in any sector. This toolkit is intended to be one of the European contributions to the field of AI. This chapter introduces the toolkit with its main components and complementary tools, providing a clear view to facilitate and encourage its adoption and wide use by the European community of developers of AI-based solutions and data scientists working in the healthcare sector and others.

Keywords Hybrid big data HPC architectures · High performance data analytics · Hardware-specific capabilities for big data GPUs FPGAs · Performance for large-scale processing

1 Context: The European AI and HPC Landscape and the DeepHealth Project

The rapid progress of different technologies is taking place within a virtuous circle that emerged thanks to the synergies between such technologies and has brought us three important advances in recent years, namely, the increase in storage capacity at a reduced price, the increase in data transmission speed and the increase in computing power provided by High-Performance Computing (HPC) and hardware accelerators. These three advances, in combination with the availability of large-enough volumes of data, have considerably boosted the growth and development of Artificial Intelligence (AI) in recent years. Mainly, thanks to the fact that the techniques of Machine Learning (ML), able to learn from data, have reached a good level of maturity and are improving the best results obtained by expert systems at the core of knowledge-based solutions in most application domains. Machine Learning is one of the most important areas of AI, which in turn includes Deep Learning (DL). As such, descriptive/predictive/prescriptive models based on AI/ML/DL techniques are becoming key components of applications and systems deployed in real scenarios for solving problems in a wide variety of sectors (e.g., manufacturing, agriculture and food, Earth sciences, retail, fintech and smart cities, among others). Nevertheless, its use in the health sector is still far from being widely spread (see [1]).

In this scenario, the European Union (EU) is fostering strategic actions to position the EU as a big worldwide player in AI, HPC and Big Data, capable of creating and deploying solutions based on cutting-edge technologies. The

DeepHealth project whose title is "Deep-Learning and HPC to Boost Biomedical Applications for Health" [2], funded by the EC under the topic ICT-11-2018-2019 "HPC and Big Data enabled Large-scale Test-beds and Applications", is one of the innovation actions supported by the EU to boost AI and HPC leadership and promote large-scale pilots. DeepHealth is a 3-year project, kicked off in January 2019 and scheduled to conclude in December 2021. DeepHealth aims to foster the use of technology in the Healthcare sector by reducing the current gap between the availability of mature-enough AI-based medical imaging solutions and their deployment in real scenarios. The main goal of the DeepHealth project is to put HPC power at the service of biomedical applications that require the analysis of large and complex biomedical datasets and apply DL and Computer Vision (CV) techniques to support new and more efficient ways of diagnosis, monitoring and treatment of diseases.

Following this aim, one of the main outcomes of DeepHealth addressing industry needs is the DeepHealth toolkit, a free and open-source software designed to be a European unified framework to offer DL and CV capabilities completely adapted to exploit underlying heterogeneous HPC, Big Data and cloud architectures. The DeepHealth toolkit is aimed at computer and data scientists as well as to developers of AI-based solutions working in any sector. It is a piece of software ready to be integrated into any software platform, designed to facilitate the development and deployment of new applications for specific problems. Within the framework of the DeepHealth project, the toolkit is being developed, tested and validated by using it to implement descriptive/predictive models for 14 healthcare use cases; nevertheless, its usefulness goes beyond the health sector, being applicable, as said, to any application domain or industrial sector. Thanks to all its features, which will be detailed throughout this chapter, the DeepHealth toolkit is technology made in EU that contributes to the development of AI in Europe.

This chapter is aligned with the technical priorities of Data Processing Architectures and Data Analytics of the European Big Data Value Strategic Research and Innovation Agenda [3]. It addresses the vertical concern Engineering and DevOps of the BDV Technical Reference Model, and the horizontal concerns Data Analytics and Data Processing Architectures focusing on the Cloud and HPC. And this chapter also relates to the *Systems, Methodologies, Hardware and Tools* cross-sectorial technology enablers of the AI, Data and Robotics Strategic Research, Innovation and Deployment Agenda [4].

This chapter also introduces the toolkit, its functionalities and its enabling capabilities with the objective to bring it closer to potential users in both industry and academia. To do so, the authors present the toolkit, its components, the adaptations that allow exploiting HPC and cloud computing infrastructures thanks to complementary HPC frameworks, and describe practical aspects to guide on its use and how to effectively integrate it for the development of AI-based applications.

2 A General Overview of the DeepHealth Toolkit

The DeepHealth toolkit is a general-purpose deep learning framework, including image processing and computer vision functionalities, enabled to exploit HPC and cloud infrastructures for running parallel/distributed training and inference processes. All the components of the toolkit are available as free and open-source software under the MIT license [5]. This framework enables data scientists to design and train predictive models based on deep neural networks, and developers to easily integrate the predictive models into existing software applications/platforms in order to quickly build and deploy AI-based solutions (e.g., support decision tools for diagnosis).

The toolkit is specifically designed to cope with big and constantly growing datasets (e.g., medical imaging datasets). Large-enough datasets enable the use of more complex neural networks and drive to improve both the accuracy and robustness of predictive models, but at the cost of dramatically increasing the demand of computing power. To do so, the DeepHealth toolkit incorporates, in a transparent manner, the most advanced parallel programming models to exploit the parallel performance capabilities of HPC and cloud infrastructures, featuring different acceleration technologies such as symmetric multi-processors (SMPs), graphic processing units (GPUs) and field-programmable gate arrays (FPGAs). It also integrates additional frameworks (i.e., COMPSs [6] and StreamFlow [7]) that allow to exploit specialized infrastructures, enabling parallelization mechanisms at different levels. Moreover, the toolkit provides functionalities to be used for both training and inference, addressing the complexity of the different available computational resources and target architectures at both the training and inference stages. Training is performed by AI experts, commonly in research-focused environments, using specialized HPC architectures equipped with FPGAs and GPUs; the goal is to maximize the number of samples processed per second keeping the overall accuracy. Inference is done with trained models in production environments (even using small devices in the edge), where the response time for predicting single samples is crucial.

The core of the toolkit consists of two libraries, namely the European Computer Vision Library (ECVL) and the European Distributed Deep Learning Library (EDDL), that are accompanied by the back end and the front end, two components to allow and facilitate the use of the libraries. The back end is a software-as-a-service module that offers a RESTful API to give access to all the functionalities of both libraries and provides independency from the programming language. The front end is a web-based graphical user interface, mainly oriented to be used by data scientists, for designing, training and testing deep neural networks. Both libraries are implemented in C++ and include a Python API to facilitate the development of client applications and integration with the wide array of Python-based data analysis libraries. Figure 1 depicts the components of the toolkit and highlights the two possible alternatives for developers of domain-specific applications to use the functionalities provided by both libraries: (1) the use of a RESTful API provided

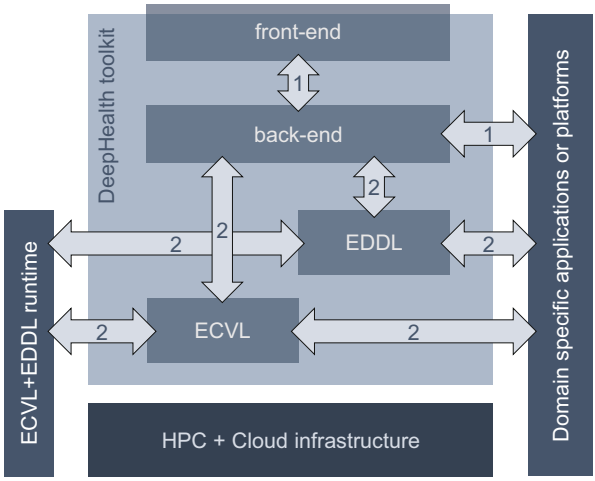

Fig. 1 Components of the DeepHealth toolkit and the two possible alternatives of interacting with the libraries. One through the back end using a RESTful API, and another using the API of both libraries. The execution of training and inference procedures over HPC + cloud infrastructures is performed by the runtime. The runtime includes adaptations to HPC frameworks ready to be executed under the control of resource managers

by the back end (represented in Fig. 1 by the arrows labelled 1) or (2) the use of the specific APIs (C++ or Python) of each library (represented in Fig. 1 by the arrows labelled 2). This second option is not independent of the programming language, yet it provides more control and flexibility over the software at the cost of additional programming complexity. The former option requires less effort from platform developers and makes applications independent of specific versions of the libraries. Only versions including changes in the RESTful API will require updating and recompiling applications. It can also be observed from Fig. 1 that all the DL and CV functionalities can be thoroughly used via the front end or via a software application/platform, in this last case with or without the back end. Additionally, Fig. 1 shows the runtime of both libraries, which can be used to launch distributed learning processes to train models on HPC and cloud architectures. Both libraries are designed to run under the control of HPC-specific workflow managers such as COMPSs [6] and StreamFlow [7], presented in Sect. 6. Once the trained models are tested and validated, they are ready to be used in production environments to perform inference from new samples by using the software applications/platforms in which libraries are integrated.

The following describes the typical workflow of the usage of the toolkit by a development team who is requested to address a new use case in a real scenario, considering that the libraries of the DeepHealth toolkit are already integrated in the platform any company developed to deploy AI-based solutions in the health sector. First, (i) data scientists, members of the team, prepare the dataset by splitting it into three subsets, namely training, validation and testing subsets. Next, (ii) the team designs several artificial neural networks and (iii) launches the training processes on HPC and cloud architectures by means of the runtime of the toolkit adapted to HPC frameworks like the ones described in Sect. 6. (iv) The team evaluates the models using the validation subset, and goes back to step (ii) to redesign some models if necessary. Sometimes, the team should come back to step (i) to consider the dataset

itself with the knowledge gained from previous iterations. (v) The model that gets the best accuracy using the testing subset is selected; then (vi) computer scientists, members of the same team, configure an instance of the application with the best model and deploy the solution in a production environment.

In itself, the DeepHealth toolkit provides the following features to AI-based application developers, data scientists and ML practitioners in general:

- Increases the productivity of computer and data scientists by decreasing the time needed to design, train and test predictive models throughout the parallelization of the training operations on top of HPC and cloud infrastructures, and without the need for combining numerous tools.
- Facilitates the easy and fast development and deployment of new AI-based applications, providing in a single toolkit, ready to be integrated, the most common DL and CV functionalities with support for different operating systems. Furthermore, it allows to perform training processes outside the application/-platform installed on production environments. To use the resulting predictive models, applications/platforms only need to integrate the libraries following one of the two possible alternatives presented.
- Relaxes the need of having highly skilled AI and HPC/cloud experts. Training processes can be executed in a distributed manner in a transparent way for data/computer scientists, and applications/platforms in production environments do not need to be adapted to run distributed processes on HPC and cloud infrastructures. Therefore, data scientists and developers do not need to have a deep understanding of HPC, DL, CV, Big Data or cloud computing.

3 The European Distributed Deep Learning Library

EDDL is a general-purpose deep learning library initially developed to cover deep learning needs in healthcare use cases within the DeepHealth project. As part of the DeepHealth toolkit, EDDL is a free and open-source software available on a GitHub public repository [5]. Currently, it supports most widely used deep neural network topologies, including convolutional and sequence-to-sequence models, and is being used in different tasks like classification, semantic segmentation of images, image description, event prediction from time-series data, and machine translation. In order to be compatible with existing developments and other deep learning toolkits, the EDDL uses ONNX [8], the standard format for neural network interchange, to import and export neural networks, including both weights and topology.

EDDL provides hardware-agnostic tensor operations to facilitate the development of hardware-accelerated deep learning functionalities and the implementation of the necessary tensor operators, activation functions, regularization functions, optimization methods and all layer types (dense, convolutional and recurrent) to implement state-of-the-art neural network topologies. The EDDL exposes two APIs (C++ and Python) with functionalities belonging to two main groups: neural

network manipulation (models, layers, regularization functions, initializers) and tensor operations (creation, serialization, I/O, mathematical transformations). The neural networks section provides both high-level tools, such as fitting and evaluating the whole model, and lower-level ones that allow developers to act on individual epochs and batches, providing finer control albeit with a slight efficiency loss in the case of using the Python API, since a larger part of the program needs to be written in Python to handle loops.

EDDL is implemented in C++, and the Python API, called PyEDDL [5], has been developed to enhance the value of EDDL to the scientific community. The availability of a Python library allows to integrate EDDL functionalities with widely used scientific programming tools such as NumPy/SciPy [9] and Pandas [10]. In particular, PyEDDL supports converting between EDDL tensors and NumPy arrays, which are key to enable interoperability with other Python scientific libraries. Moreover, since PyEDDL is based on a native extension module that wraps the C++ EDDL code, users can take advantage of the simplicity and speed of development of Python without sacrificing performance, using Python as a "glue" language that ties together computationally intensive native routines. PyEDDL allows Python access to the EDDL API and, as mentioned above, adds NumPy interoperability, allowing interaction with a wide array of data sources and tools. Like the rest of the DeepHealth Toolkit, PyEDDL is released as free and open-source software and its source code is available on GitHub [5].

In relation to hardware accelerators, EDDL is ready to run on single computers using either all or a subset of the available cores, all or a subset of the available GPU cards, and coordinating the computation flow on the FPGA cards connected to a single computer. The C++ and Python APIs of the EDDL both include a function to build neural networks that creates all the data structures according to the network topology and allocates all the necessary memory; one of the parameters of the build function is an object for describing the available hardware devices the EDDL will use to run the training and inference processes. EDDL defines the concept of Computing Service to describe hardware devices. Currently, three types of computing services are defined, namely CPU, GPU and FPGA. The number of CPU cores, GPU cards or FPGA cards to be used are indicated by the Computing Service.

Any neural network topology is internally represented by means of two directed and acyclic graphs (DAGs), one for the forward step and another one for the backward step. Each DAG defines the sequence of tensor operations to perform the computation corresponding to the entire network, so that the computations corresponding to a given layer will be performed when all its input dependencies according to the DAG have been satisfied, i.e., when the output of all the layers used as input to a given one are ready. Tensor operations are performed using the hardware devices specified by means of the Computing Service provided as a parameter to the build function. On manycore CPUs, tensor operations are performed by using the Eigen library [11] and parallelized using OpenMP [12]. When using GPU or FPGA cards, the forward and backward algorithms are designed to minimize the number of memory transfers between the CPU and GPU/FPGA cards. In the

particular case of GPUs, EDDL has three modes of memory management to address the lack of memory when a given batch size does not fit in the memory of a GPU. The most efficient one tries to allocate the whole batch in the GPU memory to reduce memory transfers at the minimum, the intermediate and least-efficient modes allow to work with larger batch sizes at the cost of increasing the number of memory transfers to perform the forward and backward steps for a given batch of samples.

GPU support in EDDL is done by means of CUDA kernels developed as part of the EDDL code. As mentioned above, the use of different hardware accelerators is completely transparent to developers and programmers who use the EDDL; they only need to create the corresponding Computing Service to use all or a subset of the computational resources. Integrating NVIDIA cuDNN library in the EDDL as an alternative to CUDA kernels is in the work plan of the DeepHealth project.

Table 1 shows the performance in terms of the accuracy obtained with the test set and the time per epoch in seconds during training. The EDDL is compared with TensorFlow [13] and PyTorch [14], the two most popular DL toolkits. The Cifar10 dataset was used. It can be observed that EDDL performs similar to the other toolkits, but EDDL still needs to improve the performance on both CPUs and GPUs when using Batch Normalization and larger topologies like VGG16 and VGG19.

EDDL support for FPGA cards is quite similar to the support for GPU cards. The developer or data scientist using the EDDL simply indicates the target device to run training or inference processes by means of a Computing Service object. Although FPGAs can also be used for training, they are more appealing for inference processes, and, therefore, FPGA support has been optimized for the inference process. Depending on the trained model, FPGA cards can be directly used. This

Table 1 Benchmark to compare EDDL with TensorFlow and PyTorch using Cifar10 with and without Batch Normalization

Model	Accuracy/time	TensorFlow		PyTorch		EDDL	
		No BN	BN	No BN	BN	No BN	BN
VGG16	Test accuracy	77.4%	71.7%	77.9%	/ 76.2%	74.6%	76.4%
	GPU time per epoch	62 s	68 s	72 s	77 s	146 s	204 s
	CPU time per epoch	1313 s	1375 s	887 s	956 s	3107 s	2846 s
VGG19	Test accuracy	66.0%	59.9%	65.5%	59.7%	68.2%	61.0 s
	GPU time per epoch	76 s	81 s	120 s	126 s	190 s	260 s
	CPU time per epoch	1703 s	1809 s	1262 s	1352 s	3872 s	3838 s
RestNet18	Test accuracy	67.6%	64.0%	66.4%	65.7%	67.3%	64.8%
	GPU time per epoch	25 s	26 s	59 s	60 s	36 s	49 s
	CPU time per epoch	1234 s	1244 s	456 s	485 s	932 s	1207 s
ResNet34	Test accuracy	66.6%	66.4%	67.8%	65.5%	66.1%	60.4%
	GPU time per epoch	44 s	46 s	97 s	101 s	65 s	89 s
	CPU time per epoch	2125 s	2140 s	834 s	895 s	1674 s	2119 s
ResNet50	Test accuracy	68.4%	61.3%	68.1%	63.1%	66.4%	61.9%
	GPU time per epoch	47 s	52 s	84 s	92 s	75 s	132 s
	CPU time per epoch	1995 s	2044 s	706 s	835 s	1684 s	2622 s

is the case when the models fit on the typically lower memory resources available on FPGA devices. If models do not fit, then two options can be used. The first one is to iteratively use FPGA cards to run the complete inference process on the model, performing operations on a step-by-step manner driven by the CPU. The FPGA support has been provided to allow this operational mode. However, quantization and compression strategies can be deployed once the model has been trained. In the DeepHealth project, the FPGA kernels mostly used on the Medical sector use cases are being optimized and adapted to quantized and compressed models. In order to deploy a model on FPGAs targeting low resource constraints and high energy efficiency, the EDDL incorporates a strategy to reduce the complexity of a deep neural network. Many techniques have been proposed recently to reduce such complexity [15–17]. These approaches include the so-called pruning techniques, whose aim is to detect and remove the irrelevant parameters from a model [18]. Removing parameters from a model has a huge impact on the deployment of the trained model on FPGA cards, since the overall size of the model reduces as well as the number of operations to generate the outcome decreases and, for instance, the power consumption. This is allowed by the typically high dimensionality of these models, where sparser and more efficient solutions can be found [19]. Towards this end, in order to deploy the model on FPGA cards targeting low resource constraints and high energy efficiency, the approach used in the EDDL is to include a structured sparsity step, where as many neurons as possible are removed from the model with a negligible performance loss.

Regarding distributed learning on HPC/cloud/HPC + cloud architectures, the EDDL includes specific functions to simplify the distribution of batches when training and inference processes are run by means of HPC frameworks like COMPSs or StreamFlow. Concretely, the COMPSs framework allows to accelerate the DL training operations by dividing the training data sets across a large set of computing nodes available on HPC and cloud infrastructures, and upon which partial training operations can then be performed. To do so, EDDL allows to distribute the weights of the network from the master node to worker nodes, and to report gradients from worker nodes to the master node, both synchronously and asynchronously. The EDDL serializes networks using ONNX to transfer weights and gradients between the master node and worker nodes. The serialization includes the network topology, the weights and the bias. To facilitate distributed learning, the serialization functions implemented in the EDDL allow to select whether to include weights or gradients.

EDDL and PyEDDL code is covered by an extensive test suite and complemented by numerous usage examples in Python and C++, including network training and evaluation with different models, ONNX serialization and NumPy compatibility. To facilitate their adoption, EDDL and PyEDDL also provide extensive documentation on installation, tensor and neural network manipulation, API usage and examples [5]. The "getting started" section contains simple examples; the most advanced ones show the use of topologies like VGG16/VGG19 [20] and U-Net [21]. Concerning installation, developers can choose between installing from source code, via conda [22], and via Homebrew for Mac OS X [23]. Additionally, pre-built Docker

images with the DeepHealth toolkit components ready to be used are available on DockerHub [24] (see Sect. 6.1).

4 The European Computer Vision Library

ECVL is a general-purpose computer vision library developed to support healthcare use cases within the DeepHealth project, with the aim of facilitating the integration of existing state-of-the-art libraries such as OpenCV [25]. ECVL currently includes high-level computer vision functionalities implementing specialized and accelerated versions of algorithms commonly employed in conjunction with deep learning; functionalities that are useful for image processing tasks in any sector beyond health.

The design of ECVL is based on the concept of *Image*, which represents the core element of the entire library. It allows to store raw data, images, and videos in a multi-dimensional dense numerical single- or multi-channel tensor. Multiple types of scientific imaging data and data formats (e.g., jpeg, png, bpm, ppm, pgm, etc.) are natively supported by ECVL. Moreover, the library provides specific functionalities to handling medical data, such as DICOM, NIfTI and many proprietary Virtual Slides (VS) formats. In the case of VS, the *Image* object allows to choose the area and the resolution to be extracted from the file. The availability of a common software architecture provided by the ECVL Hardware Abstraction Layer (HAL) allows great flexibility for device differentiation (SMPs, GPUs, and FPGAs) while keeping the same user interface. This hardware-agnostic API ensures versatility, flexibility, and extensibility, simplifying the library usage and facilitating the development of distributed image analysis tasks.

The *Image* class has been designed for representing and manipulating different types of images with diverse channel configurations, providing both reading and writing functionalities for all the aforementioned data formats. Arithmetic operations between images and scalars are performed through the *Image* class. Obviously, all the classic operations for image manipulation such as rotation, resizing, mirroring and colour space change are available. Extremely optimized processing functions, like noising, blurring, contour finding [26], image skeletonization [27] and connected components labelling [28] are implemented as well. ECVL image-processing operations can be applied on-the-fly during deep neural networks training to implement data augmentation. Given the relevance of data augmentation, ECVL provides with a simple Domain-Specific Language (DSL) to facilitate the definition of transformations to be applied and their configuration parameters. A set of transformations can thus be defined for each split of a dataset (train, validation and test subsets). Augmentation can be either provided in compiled code or through the DSL and thus read from file at runtime. More details are available in [29].

Optional modules are supplied with the library and can be activated to enable additional functionalities, such as the cross-platform GUI based on wxWidgets [30], which provides simple exploration and visualization of images contained in ECVL

Image objects, and a 3D volumes visualizer to observe different slices of a CT scan from different views.

In order to ensure an efficient and straightforward mechanism to perform distributed model training, ECVL defines the DeepHealth Dataset Format (DDF): a simple and flexible YAML-based syntax [31] that allows to describe a dataset. Regardless of the task being analysed, a DDF file provides all the information required to characterize the dataset and thus performing data loading, image pre- and post-processing and model training. A detailed description of such a format can be found in [29]. Moreover, a specific module to load and parse DDF-defined datasets is implemented and exposed by the library interface.

Like EDDL, ECVL is complemented by a Python API called PyECVL [5]. In addition to simplified programming, its main advantage is the ability to integrate with other scientific programming tools, which are abundant in the Python ecosystem. This interoperability is enabled by supporting the conversion between ECVL images and NumPy arrays. Like PyEDDL, PyECVL is based on a wrapper extension module that reroutes calls to the C++ code, allowing to reap the benefits of Python development without taking a big hit on performance. PyECVL exposes ECVL functionalities to Python, including Image objects, data and colour types, arithmetic operations, image processing, image I/O, augmentations, the DeepHealth dataset parser and the ECVL-EDDL interaction layer. As discussed earlier, its support for to/from array conversion allows to process data with NumPy as well as many other scientific tools based on it.

Regarding hardware accelerators, the ECVL supports the use of GPU and FPGA cards to run the computer vision algorithms needed in training and inference processes. The implementation for GPUs has been done using CUDA kernels, while for FPGA cards it is somewhat more complicated as FPGA cards are reconfigurable devices which allow the designer to fully customize their design and to adapt it to the algorithm they need to run. This enables, for specific application domains, more power- and energy-efficient solutions than, for instance, CPUs and GPUs. The DeepHealth project advocates for the use of FPGAs as accelerator devices for the inference process. In particular, the trained models ready for production can be launched to an FPGA card by using the FPGA support provided in both the ECVL and the EDDL libraries. The use of FPGA cards is totally transparent to data scientists who use the DeepHealth toolkit. Indeed, both libraries enable the developers who use them just to indicate which type of device the application should be using. For the specificities of the ECVL library, the use of FPGAs is appealing as most computer vision algorithms (e.g., image resize, mirror) deal with pixels rather than floating point values. FPGA devices excel at integer operations and offer massive parallelism possibilities within the device.

Like other software packages of the toolkit, ECVL and PyECVL are available as free and open-source software on a public GitHub repository, including documentation, comprehensive tests, and several usage examples of both the C++ and Python APIs [5]. Examples include data augmentation usage, handling of DeepHealth datasets, interaction with EDDL/PyEDDL, image processing and I/O. The documentation includes detailed instructions to install ECVL and PyECVL

from different options as in the case of EDDL and PyEDDL. As mentioned in the EDDL section, a set of pre-built Docker images including the components of the DeepHealth toolkit are available in the Docker hub for the DeepHealth project [24].

5 The Back End and the Front End

The four components of the DeepHealth toolkit are the ECVL, the EDDL, the back end and the front end. Figure 1 shows how the back end and the front end are interconnected with the libraries. The back end is a software module where ECVL and EDDL are fully integrated, which offers a RESTful API to allow any software application or platform to access all the functionalities provided by both libraries without the need to use the C++ or Python API. Ready-to-use pre-built Docker images (see Sect. 6.1) are available, including the back end and all the other components of the toolkit, in such a way that the developers of applications/platforms do not have to worry about the installation and configuration of the DeepHealth toolkit, they only need to provision Docker containers and, obviously, programming, using their preferred programming language, the module for their application/platform that will interact with the RESTful API offered by the back end. This way, the back end enables *managed service* usage scenarios, where a potentially complex and powerful computing infrastructure (e.g., high-performance computing, cloud computing or even heterogeneous hardware) could be transparently used to run deep learning jobs without the users needing to directly interface with it.

The front end is a web-based graphical user interface that facilitates the use of all the functionalities of the libraries by interacting with the back end through the RESTful API. The front end is the component of the toolkit visible to any type of user, but it has been mainly designed for data scientists. Without going into implementation details, the main functionalities provided by the front end are: (1) creation/edition of user profiles; (2) creation/edition of projects; (3) dataset uploading; (4) dataset selection; (5) model creation/import/export/edition/selection; (6) definition of tasks (currently supported types are classification and segmentation); (7) definition of data augmentation transformations; (8) launching training/inference processes; (9) monitoring of training processes, including visualization of the evolution of different neural network related KPIs (e.g., accuracy and loss) with respect to both training and validation data subsets; and (10) model evaluation.

In the common usage of the front end, users have the option of loading from the back end any one of the available models in the set of pre-designed models, which can be already trained. Trained models can be used to perform transfer learning tasks or just to reuse the topology by resetting weights and bias before launching a new training process.

6 Complements to Leverage HPC/Cloud Infrastructures

EDDL, ECVL and the back end are designed to be deployed on HPC and cloud infrastructures to distribute the workload of training and inference processes by following the data parallelization programming paradigm. Both libraries include specific functions to enable distributed learning. The distribution of the workload on multiple worker nodes is not directly performed by ECVL and EDDL. Instead, the libraries are complemented with workflow managers like COMPSs [6] and StreamFlow [7], specially designed for HPC/cloud environments, that manage the workload distribution of training and inference processes in combination with resource managers like SLURM [32]. To leverage hybrid HPC + cloud architectures, pre-built Docker images with all the components of the DeepHealth toolkit are ready to be deployed in scalable environments orchestrated by Kubernetes [33].

The DeepHealth toolkit is being tested on multiple HPC, cloud and hybrid HPC + cloud infrastructures to validate its ability to exploit a wide variety of architectures. The infrastructures considered in the DeepHealth project are:

- The Marenostrum supercomputer, composed of 3456 computed nodes based on Intel Xeon Platinum chips, hosted at the Barcelona Supercomputing Center.
- The MANGO cluster, composed of eight interconnected FPGAs, hosted by the Technical University of Valencia (UPV).
- The *OpenDeepHealth* (ODH) platform, implemented by the University of Torino on top of a hybrid HPC + cloud infrastructure. The HPC component is a C3S OCCAM cluster composed of 46 heterogeneous nodes, also including GPU nodes (K40 or V100). The cloud component, serving multi-tenant private Kubernetes instances, is HPC4AI [34], comprising Intel Xeon Gold 80-cores computing nodes (+2000 CPU cores) with 4 GPUs per node (80 CPU cores + V100 or T4 GPUs).
- The hybrid cloud platform, composed of a Kubernetes cluster on premise (private cloud) and another cluster running in Amazon Web Services (public cloud), provided by the company TREE Technology.

The Marenostrum supercomputer and the ODH platform are similar in terms of use; both are HPC infrastructures and both are ready to hold private clouds. The hybrid cloud facilitates vertical and horizontal scalability, providing good adaptability to different situations and uses, the possibility of deploying applications and work with data that can be shared between clouds, improving the performance of the workload. The private part of the hybrid cloud can be deployed on Marenostrum and ODH, as well as in the on-premise computer cluster of any SME. On the other hand, the MANGO cluster is an FPGA-specific computing infrastructure that is being used to evaluate some use cases of the DeepHealth project.

It is worth noting that these infrastructures offer a wide range of computing environments at different levels:

- High number of CPU computing nodes on Marenostrum, multi-GPUs nodes on the ODH platform and FPGAs in the MANGO cluster.

- The private OpenStack cloud implemented in ODH (HPC4AI), and the hybrid private+public cloud provided by TREE Technology.
- Docker containers technology used on top of bare metal layer in ODH (C3S) and in cloud platforms, using orchestration tools like Kubernetes, in TREE Technology platform and ODH (HPC4AI), and StreamFlow, which is described in Sect. 6.3.

6.1 Use of Docker Images Orchestrated by Kubernetes

The hybrid cloud platform provided by TREE Technology is a computing environment that offers the possibility of combining public and private clouds, allowing the deployment of applications and work with data that can be shared between them. This solution was built using Kubernetes technology [33], a distributed container and microservice platform that orchestrates computing, networking and storage infrastructure to support user workloads.

Software containers demonstrated to provide a good way to bundle and deploy applications. However, as system complexity increases (e.g., complex multi-component software applications, multi-node clusters) running deployments become increasingly difficult. Kubernetes supports the automation of much of the work required to maintain and operate such complex services in a distributed environment. The objective of this hybrid environment is to dynamically operate in different Kubernetes clusters running on several public clouds and on-premise infrastructures. Different Kubernetes clusters can have different hardware configurations, that is, they can have different memory and CPU settings, with or without GPUs. Once the different clusters are deployed, both in public and private clouds, it is necessary to orchestrate all the resources. For this ecosystem to work properly and be able to be coupled in the global scheme, two stages must be taken into account:

- Within a multi-cloud or hybrid-cloud context, a tool is needed to facilitate management and security tasks, as this can become a highly error prone and tedious task, while resources and Kubernetes clusters grow.
- A high-level RESTful API helps to abstract the user from the infrastructure itself, simplifying and speeding up the deployment and management of the workflows. It provides functions of varying complexity, which implements functionality abstracting the user from the potentially complex configuration of the clusters (e.g., multi-cloud, hybrid cloud, etc.). The API itself can support the addition of new Kubernetes clusters both on-premise and in the cloud from any provider with the limitation of having a minimum Kubernetes version.

The proposed hybrid cloud based on Kubernetes is a complex system, and its scalability is determined by several factors, like the number and type of nodes in a pool of nodes, the number of Pods available (Pods are the minimum deployable computer units that can be created and managed in Kubernetes), the number of

services or back ends behind a service and how resources are allocated. Usually, in a public cloud, the concept of autoscaling is available, which refers to the possibility of scaling the resources of a cluster in a self-managed manner. In the DeepHealth project, the public cloud has been configured with this autoscaling option, while for the private cloud there is no scaling policy in relation to machines.

Concerning the automatic deployment of the DeepHealth toolkit in any cloud configuration, and regardless of the complexity level of the computing infrastructures that any development team of AI-based solutions may have on hand, a set of pre-built CUDA-enabled DeepHealth Docker images, including all the components of the toolkit, are ready to be used on GPU-enabled computing resources to accelerate compute-intensive operations. All the DeepHealth Docker images available in the DockerHub [24] are CUDA-enabled and provide pre-built binaries of the libraries along with all their dependencies, such that these images can be used to create Docker-ready applications. In addition, a *toolkit* image flavour is also provided to support the developers of applications/platforms directly integrating the EDDL and the ECVL, who may prefer the C++ or Python API. These Docker images are built on the *devel* flavour of the NVIDIA/CUDA images, and add a full DeepHealth build configuration to provide a ready-to-use compilation environment for applications.

For simplified scalable deployments on cloud computing resources, a Kubernetes [33] deployment of the DeepHealth toolkit, with the web service configured, has been created and made available. The deployment automatically configures a server for the DeepHealth front end and all the back-end components (i.e., web service, worker, database, job queue, and static content server) in a flexible and scalable way. In fact, once a deployment is created, the available processing capacity can be dynamically scaled using some of the standard features of Kubernetes, such as configuring the required number of worker replicas to achieve the required throughput. The Kubernetes deployment of the DeepHealth toolkit is packaged as a Helm chart for easy deployment [35]. For simpler use cases that do not have particular scalability requirements, a Docker-compose deployment is also available. This configuration cannot distribute work over multiple nodes, but it can be trivially deployed on a single node and thus is well suited for small workloads and exploratory or development work.

6.2 COMPSs

COMPSs [6] offers a portable programming environment based on a task model, whose main objective is to facilitate the parallelization of sequential source code, written in Java or Python programming languages, to run in a distributed and heterogeneous computing environment. In COMPSs, the programmer is responsible for identifying the units of parallelism (named COMPSs tasks) and the synchronization data dependencies existing among them by annotating the sequential source code. The task-based programming model of COMPSs is then supported by its

runtime system, which manages several aspects of the application execution and keeps the underlying infrastructure transparent to the programmer. This is a key feature to guarantee the portability of COMPSs applications across a wide range of computing platforms. This will allow the DeepHealth toolkit to be tested and validated within the DeepHealth project in the infrastructures enumerated above. Regarding cloud configurations, the COMPSs runtime is being adapted within the DeepHealth project to support the hybrid cloud infrastructure. COMPSs runtime interacts with the API developed by TREE Technology to deploy workers and distribute the workload on hybrid cloud architectures. The COMPSs runtime is organized as a master-worker structure:

- The Master, executed in the computing resource where the application is launched, is responsible for steering the distribution of the application and data management.
- The worker(s), co-located with the Master or in remote computing resources, are in charge of responding to task execution requests coming from the Master.

One key aspect is that the master maintains the internal representation of a COMPSs application as a Directed Acyclic Graph (DAG) to express the parallelism. Each node corresponds to a COMPSs task and edges represent data dependencies (and so potential data transfers). Based on this DAG, the runtime can automatically detect data dependencies between COMPSs tasks: as soon as a task becomes ready (i.e., when all its data dependencies are resolved), the master is in charge of distributing it among the available workers, transferring the input parameters before starting the execution. When the COMPSs task is completed, the result is either transferred to the worker in which the destination COMPSs task executes (as indicated in the DAG), or transferred to the master if a barrier synchronization call is invoked. The parallelization of the EDDL training operation has been developed with the COMPSs tasking programming model. Due to the fine grain data dependency synchronization mechanisms supported by COMPSs, two parallel training paradigms are supported: *synchronous*, in which weights are collected and aggregated at the end of each epoch, and *asynchronous*, in which weights are increasingly aggregated as soon as a partial training is completed on the corresponding data set.

COMPSs is perfectly adapted to run in environments managed by SLURM [32], an open-source resource manager widely used in High-Performance Computing data centres to manage job queues and job allocation of incoming tasks to servers. The Ecole Polytechnique Fédérale de Lausanne (EPFL) has enhanced the core version of the SLURM resource manager with novel plugins that enable energy- and performance-aware task allocation for CPU- and memory-intensive tasks in order to increase the efficiency (in terms of performance per watt) of multiple tasks when running simultaneously on the same server and cluster. EPFL do so by proposing the use of graph-based techniques and reinforcement learning, which are low overhead and do not impact the execution time of applications. SLURM can interact with COMPSs in order to launch multiple instances of applications in a coordinated

way in an HPC infrastructure, creating a separation of concerns between resource managers, while still working in a coordinated way.

6.3 StreamFlow

StreamFlow is a novel Workflow Management System (WMS) explicitly designed in the DeepHealth project, supporting AI pipeline design and execution in different execution environments, including hybrid HPC + cloud and multi-cloud infrastructures. The portability of AI pipelines on critical data across different infrastructures is crucial for the sustainability of DeepHealth foreground technologies. To address this issue, *OpenDeepHealth* embraces StreamFlow. The ability of StreamFlow to handle sequences of computational steps makes it possible to describe a complex application as a workflow and annotate each step with an execution plan potentially targeting different nodes, e.g., selecting GPU nodes when needed, spawning across multiple sites—e.g., allowing transparent access to OCCAM and HPC4AI clusters. The idea behind this approach is that the ability to deal with hybrid workflows (i.e., to coordinate tasks running on different execution environments) can be a crucial aspect for performance optimization when working with massive amounts of input data and different needs in computational steps. Accelerators like GPUs and, in turn, different infrastructures like HPC and clouds, can be used more efficiently by selecting the execution plan that best suits the specific computational needs of each ML application developed in the project.

The StreamFlow framework is a container-native WMS written in Python. It has been designed to explore the potential benefits deriving from waiving two common properties of existing WMSs that can prevent them from fully exploiting the potential of containerization technologies. Instead of forcing a one-to-one mapping between workflow steps and Docker containers, StreamFlow allows the execution of tasks in potentially complex, multi-container environments. This allows support for concurrent execution of multiple communicating tasks in a multi-agent ecosystem, e.g., a SPMD application implemented with MPI or a COMPSs-based distributed training. StreamFlow relaxes the requirement of a single shared data space among all the worker nodes, allowing to spread different steps of a single workflow on multiple, potentially federated architectures without forcing direct communication channels among them. Moreover, StreamFlow clearly separates the definition of the AI pipeline, described as a declarative workflow, from the description of the runtime environment in charge of executing it, enforcing a separation of concerns. This allows taking advantage of using the most efficient infrastructures for the specific purpose of complex AI pipelines without burdening the AI experts with the configuration and management complexity of such infrastructures. At a very high level, an AI pipeline can comprise a training step, which usually requires very high computational power and distributed programming techniques to handle huge datasets, and an inference step, in which a fully trained model should be directly reachable from one or more user applications. StreamFlow can orchestrate the

execution of the AI pipeline, targeting the training step on HPC facilities, e.g., by using EDDL with COMPSs distributed runtime, and the inference step on the cloud cluster, e.g., by leveraging EDDL Docker containers deployed on a Kubernetes infrastructure.

7 Conclusions

The DeepHealth toolkit is presented here as a new and emerging software framework that provides European industry and research institutions with deep learning and computer vision functionalities. To cope with huge and constantly growing data sets, the toolkit has been designed to leverage hybrid and heterogeneous HPC + cloud architectures in which either all or some of the worker nodes are equipped with hardware accelerators (e.g., GPUs, FPGAs). The distributed execution of learning and inference processes is done by the runtime of the DeepHealth toolkit in a transparent manner to the common user, i.e., computer and data scientists who do not need a deep understanding of parallel programming, HPC, deep learning or cloud architectures.

The two libraries at the core of the toolkit can be easily integrated into existing software applications/platforms that European companies (SMEs and large industry) have developed to deploy AI-based solutions in any sector (e.g., decision support systems that clinicians can use to diagnose), and can be used to boost the development of new platforms and solutions. All the components of the toolkit are free and open-source software available on public repositories.

In order to foster the use of the DeepHealth toolkit, the authors have introduced the potential user to all the toolkit components and how to integrate the libraries in existing or new software applications. It is worth mentioning that, thanks to pre-built Docker images including all components with all dependencies satisfied, data scientists only need to provision Docker containers according to their needs.

The toolkit constitutes a contribution from Europe in Artificial Intelligence and smart big data analytics. Besides all the features introduced in this chapter (free and open-source framework, easy to use, portable to different architectures, wide application scope), it contributes to reducing the bottlenecks in turning AI into an enabling technology for Science (e.g., provides a way to reduce the complexity of numerical methods used in scientific environments), bringing closer the separate worlds of AI and HPC. Furthermore, it is expected to boost the adoption of AI and HPC technologies by the industry. The toolkit paves the way towards the offering of AI coupled with HPC as a service, which could be a game changer aspect in order to reach a greater number of companies. On the one hand, it offers improvements for companies that only have temporary needs for high-performance computing resources, which will be able to improve their productivity by developing their own AI solutions, and on the other hand, it could unlock the development of novel applications that need to run computationally intensive processes regularly.

Acknowledgments This chapter describes work undertaken in the context of the DeepHealth project, "Deep-Learning and HPC to Boost Biomedical Applications for Health", which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

References

1. EIT Health; McKinsey & Company, 2020 [Online]. Accessed November 11, 2020, from https://eithealth.eu/wp-content/uploads/2020/03/EIT-Health-and-McKinsey_Transforming-Healthcare-with-AI.pdf
2. The DeepHealth project [Online]. Accessed October 28, 2020, from https://deephealth-project.eu
3. Zillner, S., Curry, E., Metzger, A., Auer, S., & Seidl, R. (2017). *European big data value strategic research & innovation agenda*. Big Data Value Association.
4. Zillner, S., Bisset, D., Milano, M., Curry, E., García Robles, A., Hahn, T., Irgens, M., Lafrenz, R., Liepert, B., O'Sullivan, B., & Smeulders, A. (2020). Strategic research, innovation and deployment agenda – AI, data and robotics partnership. Third Release, 9 2020. [Online]. Accessed November 30, 2020, from https://ai-data-robotics-partnership.eu/wp-content/uploads/2020/09/AI-Data-Robotics-Partnership-SRIDA-V3.0.pdf
5. The DeepHealth GitHub Organisation [Online]. Accessed January 5, 2021, from https://github.com/deephealthproject
6. Lordan, F., Tejedor, E., Ejarque, J., Rafanell, R., Álvarez, J., Marozzo, F., Lezzi, D., Sirvent, R., Talia, D., & Badia, R. M. (2014). ServiceSs: An interoperable programming framework for the cloud. *Journal of Grid Computing, 12*(1), 67–91.
7. Colonnelli, I., Cantalupo, B., Merelli, I., & Aldinucci, M. (2020). StreamFlow: Cross-breeding cloud with HPC. *IEEE Transactions on Emerging Topics in Computing*.
8. Open Neural Network Exchange. The open standard for machine learning interoperability [Online]. Accessed October 31, 2020, from https://onnx.ai/
9. Harris, C. R., et al. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362.
10. McKinney, W., et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Austin, TX.
11. Guennebaud, G., Jacob, B., et al. (2010). Eigen v3 [Online]. Accessed November 2, 2020, from http://eigen.tuxfamily.org
12. Dagum, L., & Menon, R. (1998). OpenMP: An industry standard API for shared-memory programming. *Computational Science & Engineering, 5*(1), 46–55.
13. Abadi, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.
14. Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*.
15. Tartaglione, E., Lepsøy, S., Fiandrotti, A., & Francini, G. (2018). Learning sparse neural networks via sensitivity-driven regularization. In *Advances in neural information processing systems*.
16. Frankle, J., & Carbin, M. (2019). *The lottery ticket hypothesis: Finding sparse, trainable neural networks*.
17. Molchanov, D., Ashukha, A., & Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. In *34th International Conference on Machine Learning, ICML*.
18. Tartaglione, M., Bragagnolo, A., & Grangetto, M. (2020). Pruning artificial neural networks: A way to find well-generalizing, high-entropy sharp minima. In *Artificial neural networks and machine learning – ICANN2020*.
19. Tartaglione, E., & Grangetto, M. (2019). Take a ramble into solution spaces for classification problems in neural networks. In *International conference on image analysis and processing*.

20. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
21. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In International conference on medical image computing and computer-assisted intervention.
22. Anaconda Software Distribution. Computer software. [Online]. Accessed November 2, 2020, from https://anaconda.com; https://docs.conda.io
23. Homebrew. The Missing Package Manager for macOS (or Linux) [Online]. Accessed October 31, 2020, from https://brew.sh
24. Docker hub with DeepHealth images [Online]. Accessed October 28, 2020, from https://hub.docker.com/orgs/dhealth
25. Open Source Computer Vision Library (OpenCV) [Online]. Accessed November 3, 2020, from https://github.com/itseez/opencv
26. Allegretti, S., Bolelli, F., & Grana, C. (2020). A warp speed chain-code algorithm based on binary decision trees. In *4th International Conference on Imaging, Vision & Pattern Recognition*.
27. Bolelli, F., & Grana, C. (2019). Improving the performance of thinning algorithms with directed rooted acyclic graphs. In *International conference on image analysis and processing*.
28. Bolelli, F., Allegretti, S., Baraldi, L., & Grana, C. (2019). Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling. *IEEE Transactions on Image Processing, 29*(1), 1999–2012.
29. Cancilla, M., et al. (2021). The DeepHealth toolkit: A unified framework to boost biomedical applications. In *25th IEEE International Conference on Pattern Recognition*, 2021.
30. wxWidgets: Cross-Platform GUI Library [Online]. Accessed November 2, 2020, from https://www.wxwidgets.org
31. YAML Ain't Markup Language [Online]. Accessed November 2, 2020, from https://yaml.org/
32. Yoo, A., Jette, M., & Grondona, M. (2003). Slurm: Simple linux utility for resource management. In *Job scheduling strategies for parallel processing*. Berlin.
33. Kubernetes [Online]. Accessed November 11, 2020, from https://kubernetes.io
34. Aldinucci, M., et al. (2018). HPC4AI, an AI-on-demand federated platform endeavour. In *15th ACM International Conference on Computing Frontiers*, Ischia, 2018.
35. DeepHealth Helm chart repository [Online]. Accessed November 12, 2020, from https://deephealthproject.github.io/helm-charts/index.yaml