



DEEPHEALTH

D3.1 ECVL Library

Project ref. no.	H2020-ICT-11-2018-2019 GA No. 825111
Project title	Deep-Learning and HPC to Boost Biomedical Applications for Health
Duration of the project	1-01-2019 – 31-12-2021 (36 months)
WP/Task:	WP3/ T3.1, T3.5
Dissemination level:	PUBLIC
Document due Date:	31/05/2020 (M17)
Actual date of delivery:	08/06/2020 (M17)
Leader of this deliverable	UNIMORE
Author(s)	Costantino Grana, Federico Bolelli Michele Cancilla, Laura Canalini, Stefano Allegretti
Version	1.0



Document history

Version	Date	Document history/approvals
0.1	26/05/2020	First draft contents
0.2	03/06/2020	Corrections made after internal review
0.3	08/06/2020	Corrections after a second internal review
1.0	08/06/2020	Definitive

DISCLAIMER

This document reflects only the author's views and the European Community is not responsible for any use that may be made of the information it contains.

Copyright

© Copyright 2019 the DEEPHEALTH Consortium

This work is licensed under the Creative Commons License "BY-NC-SA".



Table of contents

Document history	2
Table of contents	3
Executive summary	4
1 Introduction	5
1.1 ECVL development	5
1.2 Continuous Integration	6
1.3 PyECVL development	6
2 ECVL environment	10
2.1 EDDL–ECVL pipeline	10
2.2 Backend	11
3 Conclusion	12

Executive summary

In this document we briefly describe the development of European Computer Vision Library (*ECVL*), in order to make a quick overview of the software which can be found publicly available in the corresponding GitHub repository (github.com/deephealthproject/ecvl). The library is the result of the work carried out in tasks T3.1 and T3.5 of WP3.

The current development of the library covers the most important scheduled CPU-side features: image reading/writing, image manipulation, integration to/from EDDLL and support for image augmentations. Moreover, a very effective hardware abstraction layer strategy has been integrated into ECVL, allowing the implementation of GPU and FPGA functionality.

The rest of the document is organized as follows: Section 1 summarizes the objectives of the ECVL and its development. In Section 2 we describe some standalone projects which use ECVL. Finally, Section 3 gives a brief evaluation of the current development.

Listing 1: ECVL Hardware Abstraction Layer

```
class HardwareAbstractionLayer
{
public:
    static HardwareAbstractionLayer* Factory(Device dev, bool shallow = false);

    virtual uint8_t* MemAllocate(size_t nbytes) = 0;
    virtual void MemDeallocate(uint8_t* data) = 0;
    virtual uint8_t* MemCopy(uint8_t* dst, const uint8_t* src, size_t nbytes) = 0;
    virtual uint8_t* MemAllocateAndCopy(size_t nbytes, const uint8_t* src)
    {
        return MemCopy(MemAllocate(nbytes), src, nbytes);
    }
    ...
    virtual void ResizeDim(const Image& src, Image& dst, const std::vector<int>& newdims, InterpolationType interp)
    { ECVL_ERROR_NOT_IMPLEMENTED }
    virtual void ResizeScale(const Image& src, Image& dst, const std::vector<double>& scales, InterpolationType
        interp) { ECVL_ERROR_NOT_IMPLEMENTED }
    virtual void Flip2D(const Image& src, Image& dst) { ECVL_ERROR_NOT_IMPLEMENTED }
    ...
}
```

1 Introduction

The main objective of the European Computer Vision Library (ECVL) is to facilitate the integration and exchange of data between existing Computer Vision (CV) and image processing libraries, while also providing new high-level Computer Vision functionality thanks to specialized/accelerated versions of some CV algorithms commonly used in combination with Deep Learning (DL) algorithms. ECVL algorithms will also be adapted to hardware accelerators.

The library will provide support for multiple operating systems and provide multiple types of scientific imaging data and data formats, with particular reference to medical imaging data formats. The fundamental importance of the library will be the availability of a common infrastructure that will allow the development of distributed image analysis activities.

The design of ECVL takes into account the objective to allow easy integration and exchange of data between existing cutting-edge libraries and their interconnection with EDDL. Indeed, the Image object, which represents the core of the entire library, has been designed to provide an easy interfacing between EDDL Tensor and ECVL Image.

Moreover, the library includes some of the computer vision algorithms which are most commonly employed in conjunction with deep learning algorithms, in order to provide specialized/accelerated versions for use with EDDL.

The source code of the library is publicly available at github.com/deephealthproject/ecvl.

This document will not include documentations about ECVL and PyECVL due to their lengthening. Anyway, the documentations are accessible at deephealthproject.github.io/ecvl and deephealthproject.github.io/pyecvl, where the ECVL one offers the browsing of both releases and developing documentations.

1.1 ECVL development

In order to guarantee optimal performance ECVL has been developed with C++ programming language, exploiting the recent features offered by the C++ 17 standard. The Image class that is the generic tensor model chosen provides a simple but effective hardware abstraction layer (HAL). In fact, if the platform employed supports devices like GPU or FPGA, ECVL can also run on these devices.

The extracted chunk of code in Listing 1 shows that HardwareAbstractionLayer class uses generic functions for managing memory or applying functions allowing a great flexibility for devices differentiation.

Moreover, the list of functions developed and their status can be monitored at github.com/deephealthproject/ecvl/blob/master/PROGRESS.md, while Figure 3 displays a snapshots of the web page taken at M17.

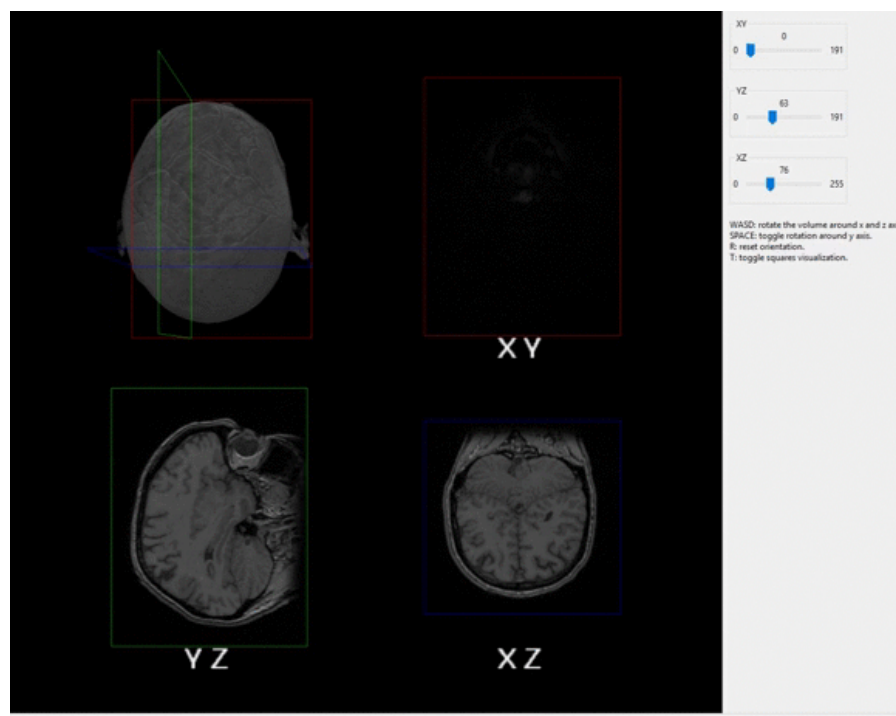


Figure 1: ECVL 3D Slices Visualizer

ECVL has been designed to hold different kind of images with diverse channels configurations. This Image attribute can contain values like 'x', 'y', and 'z' for horizontal, vertical, and depth spatial dimension or 'c' for color dimension, 't' for temporal one or, finally, 'o' indicates any other dimension.

In order to show the handling of 3D volumes, such as CT scans, the visualizer in Figure 1 has been created. This application is portable through different operating systems and allows to observe different slices of a volume from different views.

Furthermore, Figure 2 presents an image editor, powered by ECVL with some of the core library functionality. The software exploits wxWidgets, which is a cross-platform GUI library, to allow easy imaging editing such as adjust contrast and brightness, or rotate, mirror, and flip, or threshold and negate.

The above-mentioned examples have been open-sourced and are downloadable at <https://github.com/deephealthproject/ecvl-applications>.

1.2 Continuous Integration

Continuous Integration (CI) is the practice of merging in small code changes frequently - rather than merging in a large change at the end of a development cycle. The goal is to build healthier software by developing and testing in smaller increments. The ECVL development embraces this philosophy testing the code on different systems with different C++ compilers. Figure 4 shows the matrix of building employed for the library. Two main CI platform have been employed for ECVL: an open source internal one based on *Jenkins* (jenkins-master-deephealth-unix01.ing.unimore.it) and second one run by *Travis CI* (travis-ci.com/github/deephealthproject/ecvl).

These CI platforms support the development process by automatically building and testing code changes, providing immediate feedback on the success of the change.

1.3 PyECVL development

PyECVL is the Python wrapper for ECVL. It has been generated by using pybind11, a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent Boost. Python library by David Abrahams: to minimize

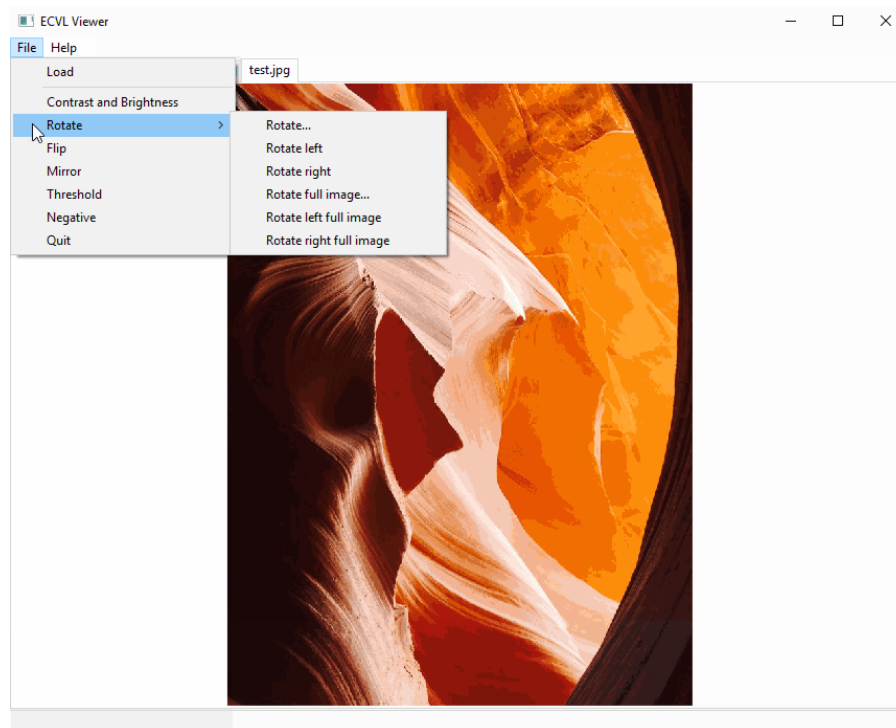


Figure 2: ECVL Imaging Editor Tool

boilerplate code in traditional extension modules by inferring type information using compile-time introspection. The following core C++ features have been mapped to Python

1. Functions accepting and returning custom data structures per value, reference, or pointer
2. Instance methods and static methods
3. Overloaded functions
4. Instance attributes and static attributes
5. Arbitrary exception types
6. Enumerations
7. Callbacks
8. Iterators and ranges

Listing 2: PyECVL Example code

```
import numpy as np
import pyecvl.ecvl as ecvl

def inc_brightness(img, rate):
    a = np.array(img, copy=False)
    max_val = np.iinfo(a.dtype).max
    a[a > max_val - rate] = max_val
    a[a <= max_val - rate] += rate

def main():
    img = ecvl.ImRead("test.jpg")
    inc_brightness(img, 10)
    ecvl.ImWrite("test_mod.jpg", img)

if __name__ == "__main__":
    main()
```

ECVL Development Status





 Implemented
  Scheduled/Work in progress
  Not implemented
  Not needed

Image Read

Functionality	CPU	GPU	FPGA
Standard Formats	✓	✗	✗
NIFTI	✓	✗	✗
DICOM	✓	✗	✗
Whole-slide image (Hamamatsu, Aperio, MIRAX, ...)	✓	✗	✗

Image Write

Functionality	CPU	GPU	FPGA
Standard Formats	✓	✗	✗
NIFTI	✓	✗	✗
DICOM	✓	✗	✗

Image Arithmetics

Functionality	CPU	GPU	FPGA
Add	✓	✗	✗
Sub	✓	✗	✗
Mul	✓	✗	✗
Div	✓	✗	✗
Neg	✓	✗	✗

Figure 3: ECVL Progress Snapshot at M17

9. Custom operators
10. Single and multiple inheritance
11. STL data structures
12. Smart pointers with reference counting like `std::shared_ptr`
13. Internal references with correct reference counting
14. C++ classes with virtual (and pure virtual) methods can be extended in Python

Listing 2 shows a short example of how ECVL can be used with Python.

The PyECVL documentation is available at deephealthproject.github.io/pyecvl.

By default, PyECVL assumes a complete ECVL installation, including optional modules (except for the GUI), and builds bindings for all of them. You can disable support for specific modules via environment variables. For instance, let us assume ECVL is installed with no OpenSlide support: by default, PyECVL will try to build the bindings for OpenSlide-specific ECVL tools and link the OpenSlide library, which might not even be present on your system. To avoid this, set the `ECVL_WITH_OPENSIDE` environment variable to `OFF` (or `FALSE`) before building PyECVL. Similarly, you can turn off DICOM and EDDL support by setting `ECVL_WITH_DICOM` and `ECVL_EDDL` to `OFF`.

Continuous integration (CPU)

Windows

OS	Compiler	OpenCV	EDDL	Infrastructure	Status
Windows 10 1903	VS 15.9.11	3.4.6	Latest Release	Jenkins	build passing

Linux

OS	Compiler	OpenCV	EDDL	Infrastructure	Status
Ubuntu 18.04.3	GCC 8.4.0	3.4.6	0.6.0	Jenkins	build passing
Ubuntu 18.04.4	GCC 6.5.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	GCC 7.5.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	GCC 8.4.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	GCC 9.3.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 5.0.2	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 6.0.1	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 7.1.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 8.0.1	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 9.0.0	3.4.10	0.6.0	Travis CI	build passing
Ubuntu 18.04.4	Clang 10.0.1	3.4.10	0.6.0	Travis CI	build passing

MacOS

OS	Compiler	OpenCV	EDDL	Infrastructure	Status
MacOSX 10.15.4	Apple Clang 11.0.3	3.4.10	0.6.0	Travis CI	build passing

Figure 4: ECVL Continuous Integration Matrix

2 ECVL environment

The development of the library produced several related projects as output. This Section lists activities which make use of ECVL and shows part of its functionality.

2.1 EDDL–ECVL pipeline

The cooperation between EDDL and ECVL is one of the main requirements of the DeepHealth project. The repository in github.com/deephealthproject/use_case_pipeline shows the full capabilities of these two libraries and their combined effort.

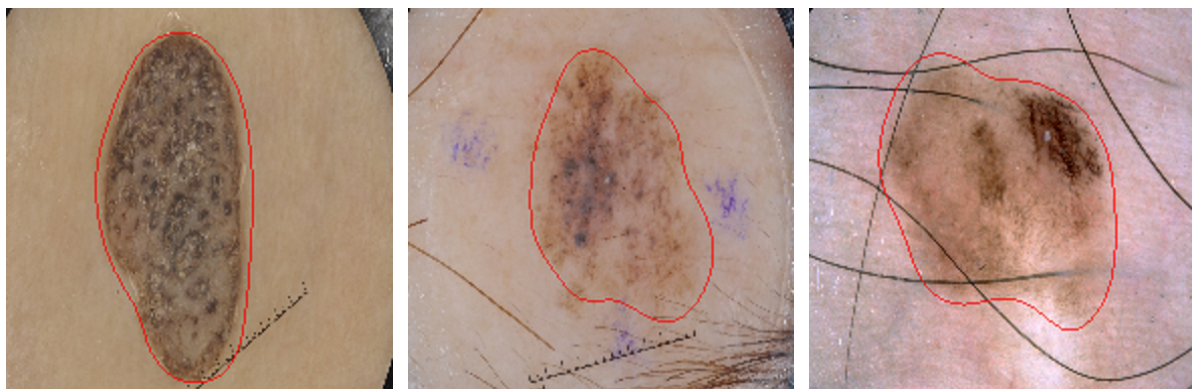
In details this repository contains four pipeline examples using EDDL and ECVL to train a Convolutional Neural Network on three different datasets (MNIST, ISIC and Pneumothorax), applying different image augmentations, for both classification and segmentation tasks.

The ISIC dataset refers to the datasets released for two different challenges:

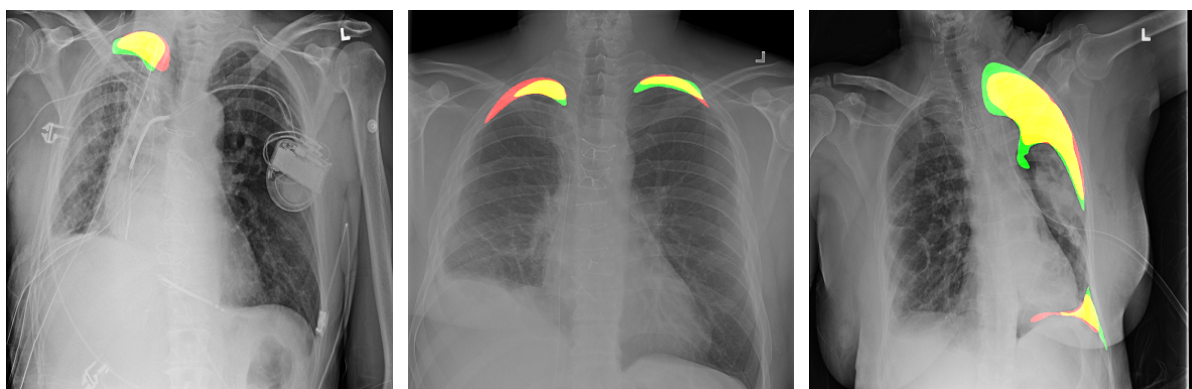
- Skin Lesion Segmentation — challenge2018.isic-archive.com
- Skin Lesion Classification — challenge2019.isic-archive.com

Furthermore, the Pneumothorax pipeline has been taken from the kaggle.com/c/siim-acr-pneumothorax-segmentation challenge hosted on *kaggle*.

The Figure 5 shows the prediction of our ISIC and Pneumothorax EDDL–ECVL pipelines. The first row (Figure 5a) shows the predicted segmentation on some samples of skin lesions dermoscopic images. On the other hand, the Figure 5b displays the recognition of pneumothorax disease on three different chest radiographic images.



(a) Red polygons, which are output of neural network, delimit the regions of interest (skin lesion).



(b) Pneumothorax segmentation masks where red indicates the prediction area, green is the ground truth, and yellow shows their intersection.

Figure 5: Output of ISIC and Pneumothorax EDDL–ECVL pipelines.

2.2 Backend

The DeepHealth toolkit (DHT), which is a core objective of the project, is composed by three main components, the two libraries ECVL and EDDL plus a front-end. However, the DHT can be thought as a software module that is divided into two parts, one visible to the user (a.k.a. graphical user interface–GUI) through a web browser (Firefox, Chrome, Edge, ...) and one invisible part (backend) that performs all the actions indicated by the user through the front-end in order to run the functionality provided by both libraries. The strength of backend, developed in T3.1 and T3.4, is that it can run on server instances which can be deployed as Linux containers defined and created using Docker and orchestrated using Kubernetes.

3 Conclusion

The development of the ECVL and its Python version PyECVL are in progress according to the work plan. The advancements have been done in strict collaboration with partners, resulting in interesting outcomes, such as the public EDDL–ECVL pipeline repository and the backend framework.

The library needs to be enlarged, for example adding more support to GPU and FPGA devices, and refined checking whether if some bottlenecks limit the global performance.