



DEEPHEALTH

EDDL

Lab 0: ECVL + EDDL environment

Winter School 24/01/2022



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Contents

What is EDDL?	4
Deep-Learning features	6
Tensor features	8
Supported architectures	10
Installation	12
Documentation	14
Resources	16





What is EDDL?



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



What is EDDL?

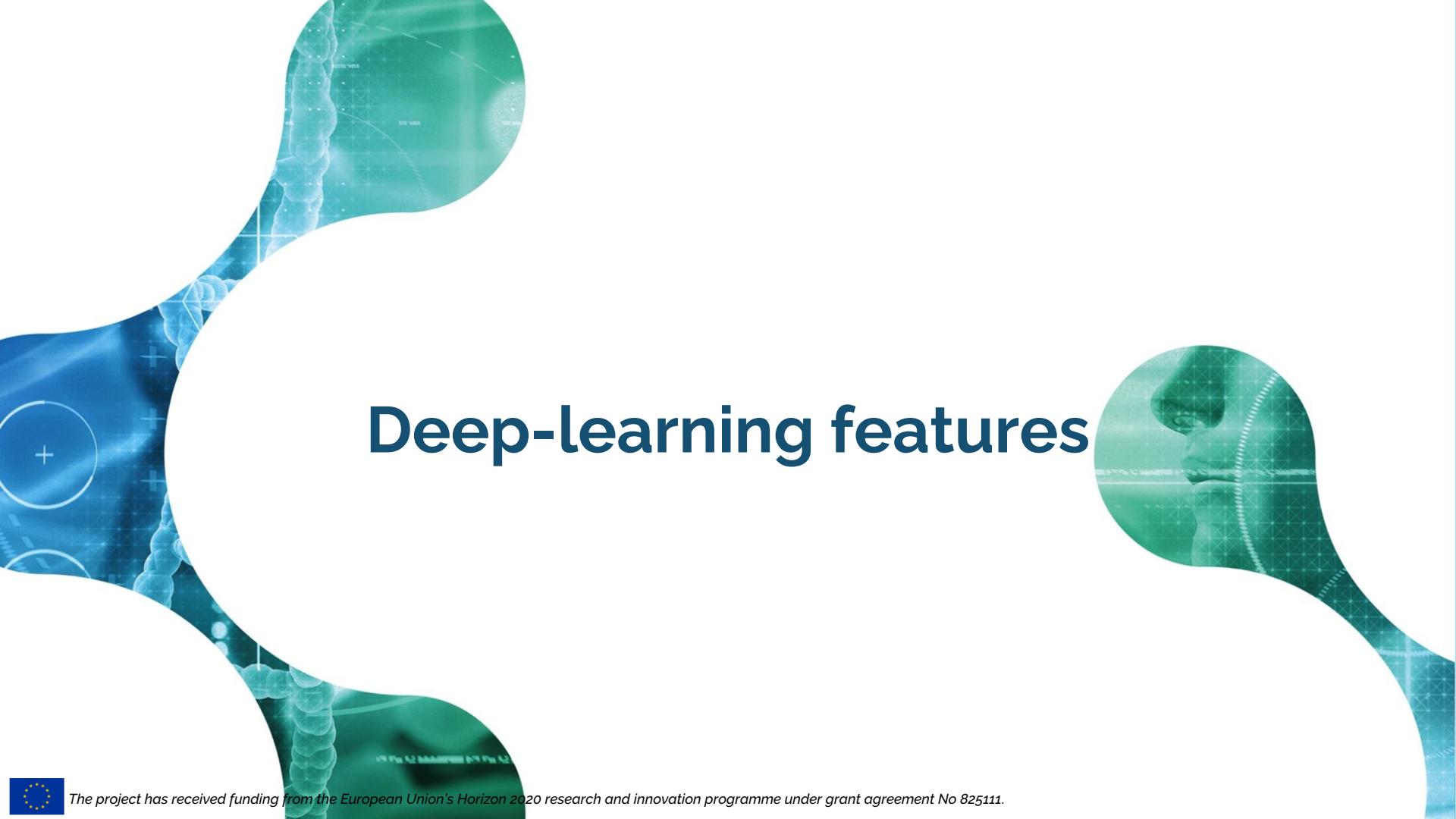
European Distributed Deep Learning Library

- Open-source library that provides two high-level features:
 - Distributed Deep Learning
 - Tensor computation
- API for C++ and Python

Key benefits

- Hardware transparency support for CPU, GPU and FPGA
- Support for Clusters, Clouds and containerized platforms
- Framework interoperability



The background of the slide features a stylized, abstract illustration of a human brain in shades of blue and green. Overlaid on the brain are various medical and scientific elements: a magnifying glass icon in the bottom left, a circular radar-like interface with a plus sign in the center, a grid pattern, and several small circular windows showing different brain slices or data sets. The overall aesthetic is modern and technological.

Deep-learning features



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Deep-learning features

Layers supported

All the main deep-learning operations are supported (See [here](#)):

- **Core layers:** Dense, Dropout, Select, Concat, Broadcasting,...
- **Activations:** ReLU, Softmax, Sigmoid,...
- **Convolutions & Poolings:** conv, pool, upsampling, transposed,... (1d, 2d, 3d)
- **Recurrent:** RNN, LSTM, GRU
- **Normalization:** BatchNorm, LayerNorm, GroupNorm,...
- **Tensor operators:** Max, Min, Sum, Mean, Equal,...
- **Data transformation & Augmentation***: Crop, Rotate, Flip, Pad, Cutout,...



Tensor features



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Tensor features

Operations supported

All the main tensor operations are supported (See [here](#)):

- **Array creation:** ones, zeros, arange, linspace, randn, clone,...
- **Array manipulation:** reshape, permute, un/squeeze, concat, stack, tile, broadcast...
- **Image manipulation:** Crop, Scale, Rotate, Pad, Flip,... (+ randoms)
- **Indexing routines:** nonzero, where, select, permute, expand,...
- **Input/Output:** load, save, load_partial,...
- **Linear Algebra:** Interpolate, Norm, Trace,
- **Logical functions:** all, any, isclose, isinf, logical_and, logical_or,...
- **Mathematical functions:** abs, cos, clamp, log, add, mult, max, min,...





Supported architectures



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Supported architectures

Working models

- **Image classification:** AlexNet, ResNet, GoogLeNet, DenseNet,...
- **Object Detection:** YOLO, RetinaNet,...
- **Object Segmentation:** U-Nets, DUC,...
- **Encoder-Decoder:** Machine translation, Image Captioning, Sentiment,...
- **Generative Adversarial Networks**



Installation



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Installation

EDDL

- **Operating system:** Windows, MacOS and Linux
- **Hardware acceleration:** CPU, GPU (cuda native/cuDNN), FPGA and COMPSs
- **Installation:**
 - **Source:** cmake
 - **Image:** Docker
 - **Package management:** conda*, brew and pip

**It will be seen in detail in the next lab session*



Documentation



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Documentation

EDDL



EDDL

Search docs

- INSTALLATION**
 - Installation
 - Build and configuration
 - Troubleshoot
 - FAQ
- GET STARTED**
 - Getting started
 - Intermediate models
 - Advanced models
- Pretained models**
 - Classify ImageNet classes with ResNet34
- VIDEOTUTORIALS**
 - Development
 - Showcase
- LAYERS**
 - Core
 - Auxiliar Layers
 - Activations
 - Data augmentation
 - Data transformation
 - Convolutions
 - Noise Layers
 - Pooling
 - Normalization
 - Merge
 - Generators
 - Operators
 - Reduce
 - Recurrent
- MODEL**
 - Model
 - Save and load ONNX models
- TRAINING**
 - Course training
 - Fine-grained training
- TEST & SCORE**
 - Test & Score
- BUNDLE**
 - Losses
 - Metrics
 - Registers
 - Initializers

Pretrained models

This short guide explains how to use pretrained models.

Classify ImageNet classes with ResNet34

In this example, we are going to classify the following image using a pretrained ResNet34 model. You can download from here.



In order to use a pretrained model, we need to know what the models expects and how we can make its output useful for us. That is, we need to know the number of inputs, their shape, the preprocessing needed, the number of outputs and the postprocessing required. Luckily, we have this information in the [verbos page](#) from which we downloaded our model.

Steps:

- Input: 3-channel RGB images of size $(H \times W \times 3)$, where H and W are expected to be 224.
- Mean and standard deviation: Images in range of $[0, 1]$ and then normalized using $\text{mean} = [0.45, 0.45, 0.45]$ and $\text{std} = [0.229, 0.229, 0.229]$.
- Output: Image scores for each of the 1000 classes of ImageNet
- Postprocessing: Apply softmax to get the probability scores for each class.

Inference

Here are going to import a new model, add a softmax layer, compiling it, perform inference on a image and print the top K predicted classes.

```
#include <vector>
#include <assert.h>
#include <eddl/eddl.h>
```

using namespace eddl;

```
#include <string>
#include <vector>
#include <assert.h>
```

using namespace std;

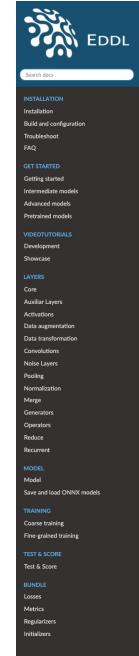
```
int main(int argc, char *argv[])
{
    // ===== SET DEFAULT VARIABLES =====
```

```
// Step 0: Load the model. The classes and the image we want to classify
string shape_file = "cifar10/classification/cifar10shape.txt";
string model_file = "models/resnet34-v2.onnx";
string img_file = "data/cifar10/cifar10_00000000000000000000000000000000.jpg";
```

```
// Step 1: Specify input
int32_t batch_size = 1;
int32_t height = 224;
int32_t width = 224;
```

```
// ===== LOAD ONNX MODEL =====
```

```
// Insert ONNX model
```



Creation Routines

Constructors

Create an uninitialized tensor

```
Tensor<T>Tensor()
```

Construct an uninitialized tensor without shape and in CPU.

```
Tensor<T>Tensor();
```

Construct an uninitialized tensor:

Parameters:

- shape – Vector of ints specifying the shape of the tensor
- dev – One of **DEV_CPU** or **DEV_GPU**

Returns: a tensor

```
Tensor<T>Tensor(int vector<int> &shape, int dev = DEV_CPU);
```

Construct tensor with initial value:

Parameters:

- data – Vector with the data to initialize the tensor with.
- shape – Vector of ints specifying the shape of the tensor
- dev – One of **DEV_CPU** or **DEV_GPU**

Returns: a tensor

```
Tensor<T>Tensor(int vector<float> &data, const vector<int> &shape, int dev = DEV_CPU);
```

Constructors & Initializers

Create tensor from generators

empty

```
Tensor<T>Tensor::empty(const vector<int> &shape, int dev = DEV_CPU);
```

Create a tensor of the specified shape filled with uninitialized data.

Parameters:

- shape – Shape of the tensor to create.
- dev – Device to use. The possible values are: **DEV_CPU** and **DEV_GPU**.

Returns: Tensor of the specified shape filled with zeros

```
Tensor<T>Tensor::empty(int vector<int> &shape, int dev = DEV_CPU);
```



Set to a constant value a region of the tensor.

Parameters:

- coords_from – Coordinates of the initial point of the crop.
- coefs_to – Coordinates of the final point of the crop.
- val – Value to fill the crop region with.

```
Tensor<T>Tensor::setCrop(int &coefsFrom[4], int &coefsTo[4], float val = 0.0f);
```

Pad

Tensor<T>Tensor::pad(const vector<int> &pads, float val = 0.0f)

Pads a tensor.

Parameters:

- pads – Padding on each border (top-bottom, left-right) or (top, right, bottom, left)
- val – Value to fill the padded region

```
Tensor<T>Tensor::pad(const vector<int> &pads, float val = 0.0f);
```

Data augmentations

Shift Random

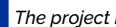
```
Tensor<T>Tensor::shift_random(const float &factor_x, const float &factor_y, WrappingMode mode = WrappingMode::Constant, float val = 0.0f);
```

Shift the tensor with a random shift value taken from a specified range. The array is shifted using spline interpolation. Points outside the boundaries of the input are filled according to the given mode.



The background of the slide features a stylized, abstract illustration of a human brain in shades of blue and green. Overlaid on the brain are various medical and scientific elements: a magnifying glass icon in the bottom left corner; a circular interface with a plus sign in the center; a grid pattern with data points and lines; and several large, semi-transparent white circles that overlap the brain image.

Resources



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



Resources

Additional links with useful information

EDDL documentation

<https://deephealthproject.github.io/eddl/index.html>

GitHub

<https://github.com/deephealthproject/eddl>

Progress

https://github.com/deephealthproject/eddl/blob/master/docs/markdown/eddl_progress.md





DEEPHEALTH

Thank you!

Salva Carrión

salcarpo@prhlt.upv.es



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.