

Using the DEEP-Hybrid-Datacloud platform

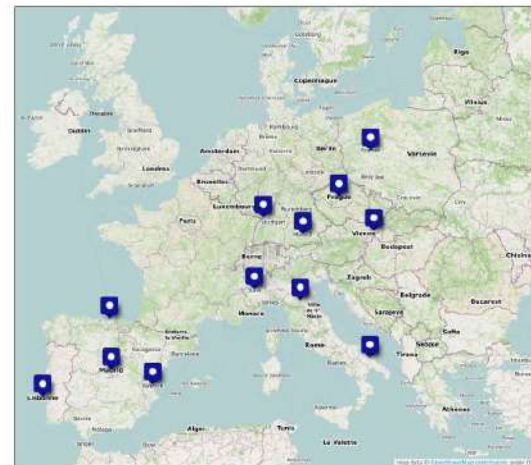
Remote webinar for iImagine
15 December 2022

Ignacio Heredia
iheredia@ifca.unican.es
Instituto de Física de Cantabria
(CSIC-UC)

Introduction - DEEP-Hybrid-Datacloud project

- The project was carried out with European Horizon 2020 funds.
- The project provides **new generation of e-infrastructures** that harness latest generation technologies, supporting deep learning and other intensive computing techniques to exploit very large data sources.
- It aims to **lower the adoption barriers** for new communities and users, satisfying the needs of both research, education communities and citizen science.

Project partners:



Introduction - AI4EOSC follow-up project

- AI4EOSC will deliver an enhanced set services for the development of AI, ML and DL models and applications in the EOSC. The services will make use of advanced features such as distributed, federated and split learning; provenance metadata; event-driven data processing services or provisioning of AI/ML/DL services based on serverless computing.

Project partners:



Introduction - The users

Basic

No machine learning knowledge. Just give me a working model to make predictions.

We offer:

- a **catalogue** full of ready-to-use modules to perform inference with your data
- an **API** to easily interact with the services
- solutions to run the inference in local or **Cloud resources**
- the ability to develop complex topologies by **composing different modules**

Intermediate

I want to retrain a working model on my personal dataset.

We offer:

- the ability to train out-of-the-box a module of the **catalogue** on your personal dataset
- an **API** to easily interact with the model
- **data storage** resources to access your dataset (DEEP-Nextcloud, OneData, ...)
- the ability to deploy the developed service on **Cloud resources**
- the ability to **share the service** with other users in the user's catalogue

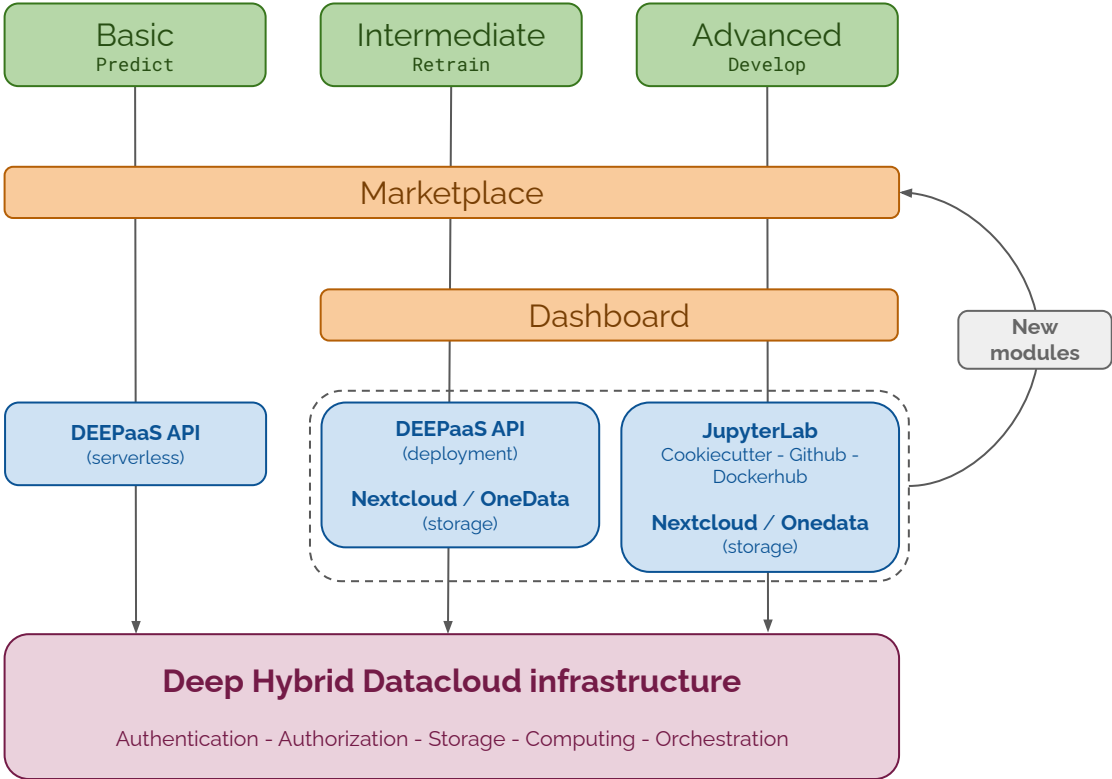
Advanced

I want to develop my custom Deep Learning model.








We offer:

- a ready-to-use environment with the **main DL frameworks** running in a dockerized solution running on different types of hardware (CPUs, GPUs, etc)
- **data storage** resources to access your dataset (DEEP-Nextcloud, OneData, ...)
- the ability to deploy the developed module on **Cloud resources**
- the ability to share the module with other users in the open **catalogue**
- the possibility to integrate your module with the **API** to enable easier user interaction

Introduction - The users



Introduction - Useful links

	Homepage	https://deep-hybrid-datacloud.eu/
	Marketplace	https://marketplace.deep-hybrid-datacloud.eu/
	Dashboard	https://train.deep-hybrid-datacloud.eu/
	Github	https://github.com/deephdc
	DockerHub	https://hub.docker.com/u/deephdc/
	Documentation	https://docs.deep-hybrid-datacloud.eu/en/latest/
	NextCloud	https://data-deep.a.incd.pt/index.php/login

(* these slides are available here)

Always refer to the docs for the most updated guidelines on using the platform.

Introduction - Webinar outline

- 1. Exploring the Marketplace**
- 2. Using the Dashboard**
 - a. Deploying a module
 - b. Making inference
 - c. Retraining a module on a new dataset
- 3. Developing a new module**
 - a. Deploying the DEEP development environment
 - b. Using the cookiecutter
 - c. Integrating it with DEEPaaS API
 - d. Adding the model to the CI pipeline
 - e. Adding the model to the Marketplace
- 4. What is next?**
 - a. New DEEPaaS features
 - b. Training Dashboard
 - c. General changes
 - d. Under-the-hood changes

1

Exploring the Marketplace

The Marketplace

Explore our marketplace!

<p>Artistic style transfer</p> <p>Model Tutorial reference (Pretrained)</p> <p>A module to apply artistic style transfer using generative adversarial networks.</p> <p>View Module</p>	<p>Bird sound classifier</p> <p>Model Tutorial reference (Pretrained)</p> <p>Classify audio recordings for species from the birdnet dataset.</p> <p>View Module</p>	<p>TF Benchmarks</p> <p>Model Tutorial</p> <p>tf.nn_benchmarks benchmark on G2T test set.</p> <p>View Module</p>
<p>Object Detection and Classification with Pytorch</p> <p>Model Tutorial reference (Pretrained)</p> <p>A neural Network Convolutional Neural Network (CNN) for object detection and classification.</p> <p>View Module</p>	<p>2D semantic segmentation</p> <p>Model Tutorial reference (Pretrained)</p> <p>2D semantic segmentation trained on the val100 dataset.</p> <p>View Module</p>	<p>Train an audio classifier</p> <p>Model Tutorial reference (Pretrained)</p> <p>Train your own audio classifier on your custom dataset. A container produced on the G2T AudioKit dataset.</p> <p>View Module</p>

2

The Dashboard

The Dashboard - Module Overview

The screenshot displays the DEEP Dashboard's Marketplace section. The interface includes a dark sidebar on the left with navigation options: Dashboard, Modules (marketplace), Deployments, and OTHER LINKS (Identity and Access, DEEP Marketplace, DEEP Documentation, Project page). The main content area features a breadcrumb trail (Home / Modules), a search bar, and a grid of 12 AI modules. Each module card contains a title, a brief description, and a list of capabilities (Model, Trainable, Inference).

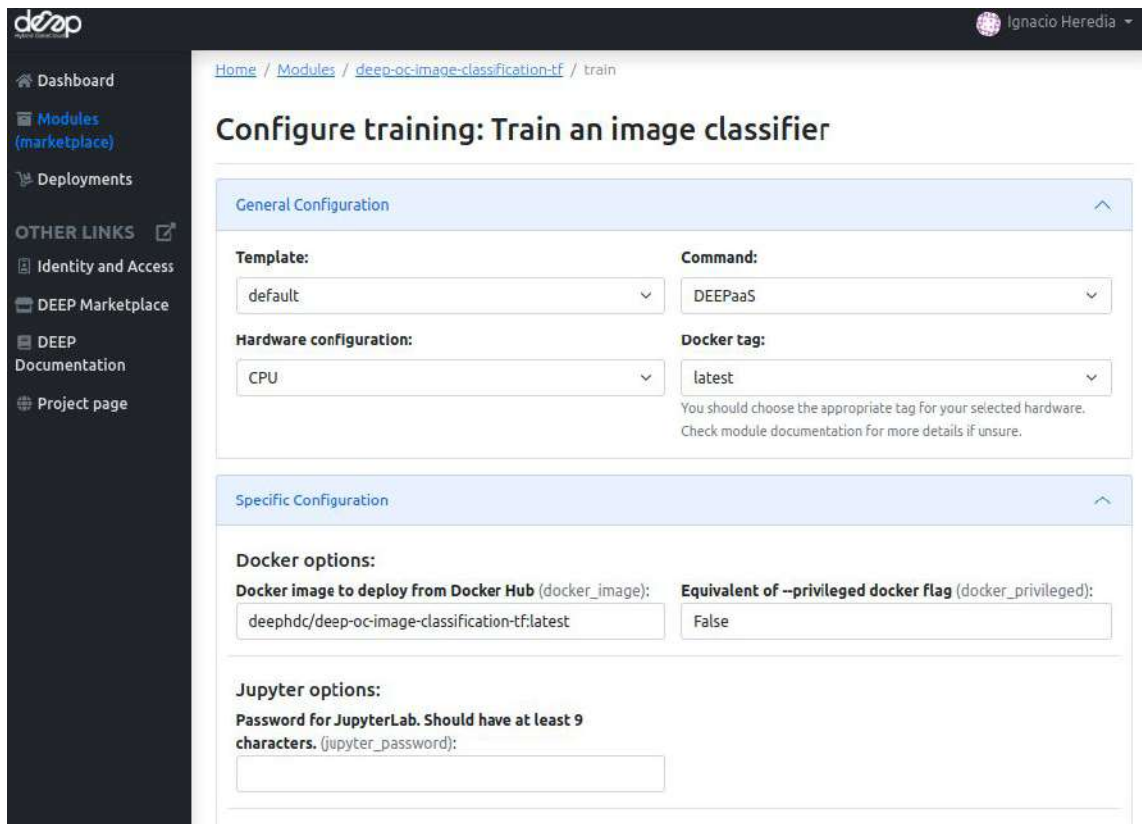
Marketplace

Home / Modules

Search...

Dogs breed detector Identify a dogs breed on the image (133 known breeds)	DEEP OC Massive Online Data Streams Deep learning for proactive network monitoring and security protection.	DEEP OC Retinopathy Test A Tensorflow model to classify Retinopathy.	Train an image classifier Train your own image classifier with your custom dataset. It comes also pretrained on the 1K ImageNet classes.
Plants species classifier Classify plant images among 10K species from the iNaturalist dataset.	Conus species classifier Classify conus images among 70 species.	Phytoplankton species classifier Classify phytoplankton images among 60 classes.	Seed species classifier Classify seeds images among 700K species.
Upscale multispectral satellites images Upscale (superresolve) low resolution bands to high resolution in multispectral satellite imagery.	Speech keywords classifier Train a speech classifier to classify audio files between different keywords.	Body pose detection Detect body poses in images.	Train an audio classifier Train your own audio classifier with your custom dataset. It comes also pretrained on the 527 AudioSet classes.

The Dashboard - Deploying a module



The screenshot shows the DEEP dashboard interface. The top navigation bar includes the DEEP logo, the user name 'Ignacio Heredia', and a dropdown menu. The left sidebar contains navigation links: Dashboard, Modules (marketplace), Deployments, OTHER LINKS (Identity and Access, DEEP Marketplace, DEEP Documentation, Project page). The main content area displays the breadcrumb 'Home / Modules / deep-oc-image-classification-tf / train' and the title 'Configure training: Train an image classifier'. The configuration is divided into two sections: 'General Configuration' and 'Specific Configuration'. The 'General Configuration' section includes dropdowns for 'Template' (default), 'Command' (DEEPaaS), 'Hardware configuration' (CPU), and 'Docker tag' (latest). A note below the Docker tag dropdown states: 'You should choose the appropriate tag for your selected hardware. Check module documentation for more details if unsure.' The 'Specific Configuration' section includes 'Docker options' with a text input for 'Docker image to deploy from Docker Hub (docker_image):' containing 'deephdc/deep-oc-image-classification-tf:latest' and a checkbox for 'Equivalent of --privileged docker flag (docker_privileged):' set to 'False'. Below this is the 'Jupyter options' section with a text input for 'Password for JupyterLab. Should have at least 9 characters. (jupyter_password):'.

Configurable options

- **docker image** (from deep-oc, but also custom docker images)
- **hardware** (#cpus, #gpus, RAM)
- **storage** (OneData, Nextcloud volumes)
- **services** (DEEPaaS, JupyterLab)

The Dashboard - Making inference

Launch `image-classification-tf` module with DEEPaaS.

POST /v2/models/imgclas/predict/ Make a prediction given the input data

Parameters

Name	Description
data file <i>(formData)</i>	Select the image you want to classify. <i>Default value : null</i>
urls string <i>(query)</i>	Select an URL of the image you want to classify. <i>Default value : null</i>
timestamp string <i>(query)</i>	Model timestamp to use for prediction. Group name: testing Choices: ['default_imagenet'] Type: str <i>Available values : "default_imagenet"</i> <i>Default value : "default_imagenet"</i>
ckpt_name string <i>(query)</i>	Checkpoint inside the timestamp to use for prediction. Group name: testing Type: str <i>Default value : "final_model.h5"</i>



Response body

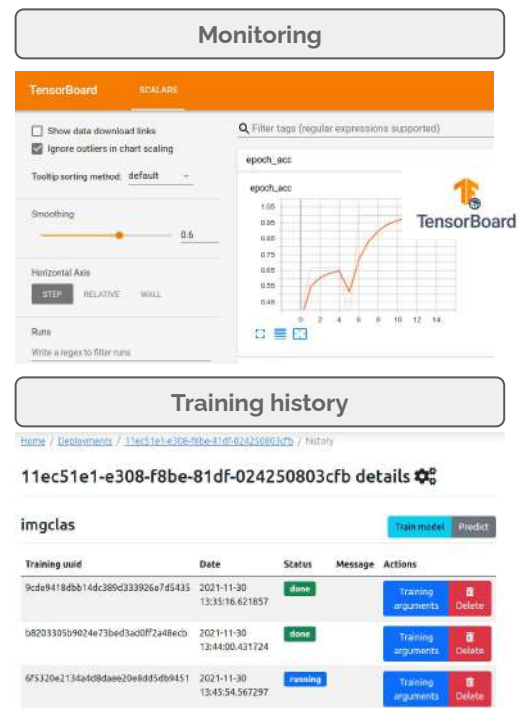
```
{
  "status": "OK",
  "predictions": {
    "labels": [
      "lion",
      "fur_coat",
      "brown_bear",
      "hyena",
      "timber_wolf"
    ],
    "probabilities": [
      0.9763001203536987,
      0.005802886560559273,
      0.00480994675308466,
      0.0014703389024361968,
      0.001060070120729506
    ]
  }
}
```

The Dashboard - Retraining a module

- 1) Launch `image-classification-tf` module with JupyterLab (remember adding password).
- 2) Copy some demo files to make a mock dataset.
- 3) Terminal: `deepaas-run --listen-ip 0.0.0.0` to launch DEEPaaS.

POST	<code>/v2/models/imgclas/train/</code>	Retrain model with available data
GET	<code>/v2/models/imgclas/train/</code>	Get a list of trainings (running or completed)
GET	<code>/v2/models/imgclas/train/{uuid}</code>	Get status of a training

A proper tutorial on how to do this with a real dataset using our remote storage will be given in **January's webinar**. In the meantime, the steps are detailed in the docs.



3

Develop your module

Developing - DEEP Development Environment

DEEP Development Environment

The DEEP Development Environment provides a ready to use JupyterLab instance that enables you to develop code using Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner.

 Create environment

Configure training: DEEP Development Environment

General Configuration

Template: default	Command: DEEPaaS
Hardware configuration: CPU	Docker tag: latest

You should choose the appropriate tag for your selected hardware. Check module documentation for more details if unsure.

Specific Configuration

Docker options: Docker image to deploy from Docker Hub (docker_image): deepcdc/deep-oc-generic-dev:latest	Equivalent of --privileged docker flag (docker_privileged): False
Jupyter options: Password for JupyterLab. Should have at least 9 characters. (jupyter_password): <input type="text"/>	

Configurable options

- **docker image** (from deep-oc, but also custom docker images). Eg:
 - **Tensorflow** docker
 - **Pytorch** docker
 - ...
- **hardware** (#cpus, #gpus, RAM)
- **storage** (OneData, Nextcloud volumes)
- **services** (DEEPaaS, **JupyterLab**)

Developing - DEEP Cookiecutter

This is the easiest way to develop any new module from scratch as it will take care of generating all the nitty-gritty details that we will cover in the following slides (entrypoints, files, Jenkinsfile, Dockerfile, etc).

- Use the command: `cookiecutter https://github.com/deephdc/cookiecutter-deep -b master`
- Answer questions:
 - Project name, description, version, license type
 - Author name, email, Github account
 - Dockerhub account, Docker base image
- This will generate two folders. Eg:
 - `mymodule`: This is where the project code is located
→ Example: <https://github.com/deephdc/image-classification-tf>
 - `DEEP-OC-mymodule`: This contains the Dockerfile of the project
→ Example: <https://github.com/deephdc/DEEP-OC-image-classification-tf>

Developing - Integrating with DEEPaaS

- Head over to `mymodule`. Any module that wants to integrate with DEEPaaS should have two minimum requirements:
 - it should define a file (eg. `mymodule/mymodule/api.py`) with the functions to interact with the module.

These functions should define:

- the model metadata
- the input args for training
- the input args for prediction
- the response structure for prediction
- the train function
- the predict function
- a model warming function for prediction

```
get_metadata()
```

```
get_train_args()
```

```
get_predict_args()
```

```
schema
```

```
train()
```

```
predict ()
```

```
warm()
```

→ Minimal example: https://github.com/deephdc/demo_app/blob/master/demo_app/api.py

→ Full example: <https://github.com/deephdc/image-classification-tf/blob/master/imgclas/api.py>

- it should define an entrypoint in `mymodule/setup.cfg` pointing to that file

→ Example: https://github.com/deephdc/demo_app/blob/master/setup.cfg#L25-L27

Developing - Customizing the Dockerfile



- Head over to `DEEP-0C-mymodule` and modify the Dockerfile following your needs:
 - install additional packages,
 - change the base image,
 - etc.

Developing - Continuous Integration



- Both `mymodule` and `DEEP-OC-mymodule` have their respective `Jenkinsfile` that define the actions to be taken when a change is committed to the repos.
- Typical workflows:
 - `mymodule/Jenkinsfile` will:
 - run PEP8 style analysis
 - trigger of `DEEP-OC-mymodule/Jenkinsfile`.

→ Example: <https://github.com/deephdc/image-classification-tf/blob/master/Jenkinsfile>
 - `DEEP-OC-mymodule/Jenkinsfile` will:
 - build Docker images for different branches (train/test) and different hardware (cpu/gpu)
 - upload the image to DockerHub
 - build Docker images of other dependent modules. For example, changes in the code of image-classification should rebuild all Docker images of applications that were trained with that code (plant classifier, seed classifier, etc).
 - refresh the module page in the Marketplace (see next step)

→ Example: <https://github.com/deephdc/DEEP-OC-image-classification-tf/blob/master/Jenkinsfile>

Developing - Integrating to the Marketplace

- Head over to `DEEP-OC-mymodule` and modify `metadata.json` with the info relevant to your module. This is the information that will appear in the Marketplace page.
- Make a Pull Request to add your module [here](#). This will create the Jenkins pipeline for your module and will add the module to the Marketplace and the Training Dashboard.



**Congratulations,
you're done!**



4

What is next?

What's next? - New DEEPaaS features

- Easier module integration via decorators/hints

Midterm

Before (webargs)

```
3 from webargs import fields, validate
4
5 def get_predict_args():
6     arg_dict = {
7         "demo-str": fields.Str(
8             required=False,
9             missing='some-string',
10        ),
11        "demo-int": fields.Int(
12            required=False,
13            missing=1,
14        ),
15    }
16
17
18 schema = {
19     "demo-list": fields.List(
20         fields.Float()
21     ),
22 }
23
24
25 def predict(**kwargs):
26     return {"demo-list": [1, 2, 3]}
27
```

After (type hints)

```
32 def predict("demo-str": str,
33            "demo-int": int,
34            ) -> dict:
35     return {"demo-list": [1, 2, 3]}
36
```

- Computing CO2 emissions of training runs using CodeCarbon

Midterm



What's next? - Friendlier inference UI



Before (Swagger UI)

Swagger UI for demo_app. Endpoints include GET /v2/models/demo_app/train/{uuid}, DELETE /v2/models/demo_app/train/{uuid}, and POST /v2/models/demo_app/predict/. The predict endpoint has parameters: demo-str (string, some-string), demo-str-choice (string, choice2), demo-int (integer, 1), demo-image (file, image), demo-audio (file, audio), and demo-video (file, video). An Execute button is at the bottom.

After (Gradio based)

Mature

demo_app

A minimal toy application for demo and testing purposes. We just implemented dummy inference, ie. we return the same inputs we are feed.

Inputs

Gradio-based inference UI for demo_app. It features interactive input fields for DEMO-STR (some-string), DEMO-STR-CHOICE (choice2), DEMO-INT (1), DEMO-INT-RANGE (50), DEMO-FLOAT (0.1), DEMO-BOOL (checked), DEMO-DICT ({"a": 0, "b": 1}), DEMO-LIST-OF-FLOATS (0.1, 0.2, 0.3), and DEMO-IMAGE (image with Edit button). On the right, there are preview areas for DEMO-STR, DEMO-IMAGE, DEMO-INT-RANGE, DEMO-VIDEO (video player), and DEMO-AUDIO (audio player).

What's next? - Friendlier inference UI



Before (Swagger UI)

The screenshot shows the Swagger UI for a REST API endpoint. It displays the request URL, the request body (a JSON object), and the response body (a JSON object). The response body contains the following data:

```
{
  "demo-str": "some-string",
  "demo-str-choice": "choice2",
  "demo-int": 1,
  "demo-int-range": 50,
  "demo-float": 0.1,
  "demo-bool": true,
  "demo-dict": {
    "a": 0,
    "b": 1
  },
  "demo-list-of-floats": [
    0.1,
    0.2,
    0.3
  ],
  "demo-image":
    "UkLGRjPgEABXQVZF2m10IBAAABAAIAQBGAAB9AAAEABAAZGF0YTRFEAC0/xCAD/9DAOP+4/+T/Y/c/zARAFACsAN7/VADS/0MAEQ0j/wCALP+q/0X/v/00AEAw/5/9T/3ADv/1EAtPKAAUAS/9RAFQAF/DoNlg8AMv/ZgLIAMzRZ2AD/CcC/2Z/b/+X/3/a+0KAM/PQ8dAS19b/wAgGcX//L/4+v/Y/L7X/0P9+/4X/SQAR/57/Rv/R/1f/9/Q/goA1AA1A0b/RABeA14qAARACs4v+LAGAgTgIACP/Y/ob/3/97AoaMgB1ARQoQfTADgByt++cAH4ADQC+/w1///j/mABgAM/7T/2P+Z/9v/b/9UAD0BtQ/M+/K/0CAZuBU/tv/dv+pv/7n/MADq/w4A1f/y/4L/sP84AD4A/f6AGoAwf9KADMAy//hz/zD/2v5w0/0AMN0SPD/wCAT/05/87+8Pv/ygZPMG/2DAGC2AVKAVP9ALp/mjCL/9bVx/9UAK/3/45/0oY/REAAgAumF1ATv/vZ0MRC0a8b/00ASAFj/vPp9AT0Mw9ZAGAAnt/4/QT/919t/zZ/CDDP/01AK/+Vv+oRq0IqMMAIC00A3AwTBPADZ/1N+EqBD/+P+dv93AFz/P0B2AM/Fq0//ysBqBv/01/BAFLAP/0+Ad3/2ATB4U0R6AAVRNAlvGm/zIRn0D/0VYAL0G0AAf9DP7o/2MBIv/v/kv/1/8BAET0RQCTA0F1
```

After (Gradio based)

Mature

demo_app

A minimal toy application for demo and testing purposes. We just implemented dummy inference, ie. we return the same inputs we are feed.

The screenshot shows the Gradio demo application interface. It displays a series of input fields and their corresponding outputs:

- DEMO-STR: some-string (0.736)
- DEMO-STR-CHOICE: choice2
- DEMO-INT: 1
- DEMO-INT-RANGE: 50
- DEMO-FLOAT: 0.1
- DEMO-BOOL: true
- DEMO-DICT: {"a": 0, "b": 1}
- DEMO-LIST-OF-FLOATS: [0.1, 0.2, 0.3]
- DEMO-IMAGE: A landscape image of a field with blue flowers.
- DEMO-VIDEO: A video player showing a video of the Earth from space.
- DEMO-AUDIO: An audio player showing a waveform.

Outputs

What's next? - Training dashboard

Midterm

- Organizing training run in experiments
 - hyperparameter optimization
 - easier side-by-side comparison of training runs

Midterm

- Enable VScode IDE to develop code in the Dashboard. This will make for a more pleasant coding experience compared to JupyterLab.



What's next? - General

Longterm

- Richer module metadata language, to keep track of:
 - training datasets
 - models
 - training execution pipelines

Longterm

- Run trainings in a federated manner. Data won't leave your local resources, which is useful when dealing with sensitive data.

What's next? - Under-the-hood changes

Mature

- Support Dask in DEEPaaS for better multi-platform integration (Yarn, Kubernetes, Slurm, etc).



Midterm

- Moving for Orchestrator to Nomad for managing deployments. This means it would be easier to integrate computing resources from external providers.



Longterm

- Better orchestration of pipelines using Prefect (or similar tools).



Questions



AI4

eosc



European
Commission

Horizon 2020
European Union funding
for Research & Innovation



CSIC
Spanish Council of Research

The iImagine project receives funding from the European Union Horizon Europe Programme – Grant Agreement number 101058625.
The AI4EOSC project has received funding from the European Union's Horizon Research and Innovation programme under Grant agreement No. 101058593