OpenDR —
Open Deep Learning Toolkit for Robotics

Project Start Date: 01.01.2020
Duration: 48 months
Lead contractor: Aristotle University of Thessaloniki

Deliverable D7.3: Intermediate public version of the OpenDR toolkit

Date of delivery: 31 Dec 2022

Contributing Partners: AUTH, TAU, AU, TUD, ALU-FR, CYB, PAL
Version: v4.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449.

Title	**D7.3: Intermediate public version of the OpenDR toolkit**
Project	**OpenDR** (ICT-10-2019-2020 RIA)
Nature	Open Research Data Pilot
Dissemination Level:	**PU**blic
Authors	Paraskevi Nousi (AUTH), Charalampos Symeonidis (AUTH), Pavlos Tosidis (AUTH), Nikolaos Passalis (AUTH), Efstratios Kakaletsis (AUTH), Maria Tzelepi (AUTH), Anastasios Tefas (AUTH), Nikolaos Nikolaidis (AUTH), Gizem Bozdemir (PAL), Thomas Peyrucain (PAL), Dias Daniel (CYB), Halil Ibrahim Ugurlu (AU), Illia Oleksiienko (AU), Lukas Hedegaard Morsing (AU), Negar Heidari (AU), Erdal Kayacan (AU), Alexandros Iosifidis (AU), Daniel Honerkamp (ALU-FR), Niclas Vödisch (ALU-FR), Jenni Raitoharju (TAU), Anton Muravev (TAU), Dat Thanh Tran (TAU), Jelle Luijkx (TUD), Bas van der Heijden (TUD)
Lead Beneficiary	PAL (PAL Robotics)
WP	7
Doc ID:	OPENDR_D7.3.pdf

Document History

Version	Date	Reason of change
v1.0	01/09/2022	Deliverable structure template ready
v2.0	09/12/2022	Contributions from partners finalized
v3.0	12/12/2022	Comments from internal review (Alea Scovill - AGI)
v4.0	28/12/2022	Final version ready for submission

Contents

Executive Summary

This document aims at supplementing the second public version of OpenDR toolkit released in M36. It provides details about accessing, downloading and using the toolkit for all the provided installation methods, namely: by cloning the GitHub repository, by installing it using *pip* or through the provided *docker* images. Required dependencies of each method have been specified and demo instructions to showcase how to use some of the tools in the toolkit are provided. Lastly, the document provides instructions to run the ROS/ROS2 nodes integrated into the second version of the OpenDR toolkit and how to customize it.

1 Introduction

OpenDR aims at developing an open, non-proprietary modular toolkit that can be easily used by robotics companies and research institutions to efficiently develop, evaluate and deploy AI and cognition technologies to robotics applications. At a high level, OpenDR contains a selection of cognition and perception algorithms, along with general-purpose functionalities that are necessary for common robotics tasks. This technical report (Deliverable D7.3) aims at supplementing the second public version of OpenDR toolkit released on M36. It provides details about accessing, downloading, installing and using the toolkit.

2 Changelog

Notable changes with the second version of the toolkit are summarized here.

Features:

- Added end-to-end planning tool
- Added seq2seq-nms module, along with other custom NMS implementations for 2D object detection
- Added a standalone pose-based fall detection tool
- Added continual transformer encoders
- Added ambiguity measure utility tool
- Added YOLOv5 object detection tool
- Added continual spatio-temporal graph convolutional networks tool
- Added SiamRPN 2D tracking tool
- Added facial emotion estimation tool
- Added high resolution pose estimation tool

Enhancements:

- Added support for modular pip packages allowing tools to be installed separately
- Updated toolkit to support CUDA 11.2 and improved GPU support

Bug Fixes:

- Fixed `BoundingBoxList`, `TrackingAnnotationList`, `BoundingBoxList3D` and `TrackingAnnotationList3D` confidence warnings
- Fixed undefined `image_id` and segmentation for COCO `BoundingBoxList`
- Fixed continual X3D ONNX support
- Fixed face recognition compilation error
- Fixed efficient panoptic segmentation submodule
- Updated wheel building pipeline to include missing files and removed unnecessary dependencies

- Updated dataset preparation scripts to create correct validation ground truth for efficient panoptic segmentation
- Added specific configuration files for the provided pre-trained models for efficient panoptic segmentation
- Pass key by const reference in `json_get_key_string()` in face recognition's C API
- Fixed height check in lightweight open pose
- Fixed bugs where ONNX optimization failed on specific learner parameterization in lightweight open pose

3 Accessing the OpenDR toolkit

The toolkit is developed using the well-established GitHub platform, following robust development methodologies, including continuous integration and strict code review guidelines, as described in D2.2 and D7.2. The most recent version of the toolkit can be accessed at:

https://github.com/opendr-eu/opendr

The `master` branch contains the latest stable version of the toolkit and the `develop` branch a version that includes the latest additions, refactors and module upgrades. Although CI tests will maintain stability and high quality in both branches, the `develop` one is likely to change often so it is less adapted for daily usage or production.

OpenDR provides an intuitive and easy-to-use **Python interface**, a **C API** for selected tools, a **ready-to-use ROS/ROS2 nodes** and **a wealth of usage examples and supporting tools**. OpenDR is built to support Webots Open Source Robot Simulator, while extensively following industry standards, such as ONNX model format and OpenAI Gym Interface. Detailed documentation can be found in the OpenDR repository and wiki.

4 Installing the OpenDR toolkit

To maximize the visibility and ease-of-use of the toolkit, we provide three different ways for installing the toolkit:

1. By cloning the GitHub repository
2. Using *pip*
3. Using *docker*

The first way provides a fully functional version of the toolkit that can be installed in various platforms. *pip* is a straightforward way to install and experiment with the Python API of the toolkit, while docker images are provided to experiment with toolkit functionalities in a pre-configured environment with very little effort, as well as for other containerized applications.

The following subsection provides an overview of the installation process. OpenDR is designed to be easy-to-use and install in order to maximize its impact. To this end, installation scripts have been prepared to ensure that this process will be very easy, even for novice users. Up-to-date instructions and additional details are available on OpenDR's GitHub repository.

4.1 Installation by cloning the GitHub repository

4.1.1 Installation procedure

To install the toolkit on a Linux system, please first make sure that *git* is available on the system:

```
sudo apt install git
```

Then, the toolkit should be downloaded locally:

```
git clone --depth 1 --recurse-submodules -j8  \
    https://github.com/opendr-eu/opendr
```

To install the toolkit an installation script is available:

```
cd opendr
./bin/install.sh
```

The installation script automatically installs all the required dependencies. Note that we can set the training/inference device using the `OPENDR_DEVICE` variable. The toolkit defaults to using CPU. If we want to use GPU, we can set this variable accordingly *before* running the installation script:

```
export OPENDR_DEVICE=gpu
```

The installation script creates a *virtualenv*, where the toolkit is installed. OpenDR environment can be activated similar to any other *virtualenv*:

```
source ./bin/activate.sh
```

All functionality (e.g., ROS, tools, demos, etc.) are then readily available.

4.1.2 Demo

For example, in order to run the *human gesture recognition* demo you can:

```
cd projects/python/perception/multimodal_human_centric/rgbd_hand_gesture_recognition
python3 gesture_recognition_demo.py \
    -input_rgb input_rgb.png -input_depth input_depth.png
```

where the two images mentioned are shown in Figure 1 (also available in the demo's folder). The result of the demo is *"Punch with confidence 0.606"*.

4.2 Installation using *pip*

To increase the visibility of the toolkit, PyPI packages have been prepared for each tool.

Figure 1: (left) RGB input image and (right) depth input image

4.2.1 Installation procedure

When installing the Python-API of the toolkit, it is necessary to first install the required dependencies:

```
sudo apt install python3.8-venv libfreetype6-dev git build-essential cmake \
   python3-dev wget libopenblas-dev libsndfile1 libboost-dev libeigen3-dev
python3 -m venv venv
source venv/bin/activate
pip install wheel
```

Then you can install the toolkit with:

```
pip install opendr-toolkit-engine
pip install opendr-toolkit
```

If your CPU does not support AVX2, you will need to set `export DISABLE_BCOLZ_AVX2=true` prior to installing the toolkit.

4.2.2 Enabling GPU-acceleration

The same OpenDR package is used for both CPU and GPU systems however appropriate GPU-enabled dependencies are required to use a GPU with OpenDR. In order to do so, it is needed to install the following packages prior to installing the toolkit:

```
pip install torch==1.9.0+cu111 torchvision==0.10.0+cu111 torchaudio==0.9.0 \
  -f https://download.pytorch.org/whl/torch_stable.html
pip install 'git+https://github.com/facebookresearch/detectron2.git'
pip install mxnet-cu112==1.8.0post0
```

4.2.3 Installing only a particular tool using pip

The instructions above will install the entire toolkit. It is however possible to only install a specific OpenDR tool as the package has been split to that effect. If you wish to only perform pose estimation you can:

```
pip install opendr-toolkit-engine
pip install opendr-toolkit-pose-estimation
```

Note that `opendr-toolkit-engine` must always be installed in the system. The following packages are distributed:

```
opendr-toolkit-activity-recognition
opendr-toolkit-speech-recognition
opendr-toolkit-semantic-segmentation
opendr-toolkit-skeleton-based-action-recognition
opendr-toolkit-face-recognition
opendr-toolkit-facial-expression-recognition
opendr-toolkit-panoptic-segmentation
opendr-toolkit-pose-estimation
opendr-toolkit-fall-detection
opendr-toolkit-compressive-learning
opendr-toolkit-hyperparameter-tuner
opendr-toolkit-heart-anomaly-detection
opendr-toolkit-human-model-generation
opendr-toolkit-multimodal-human-centric
opendr-toolkit-object-detection-2d
opendr-toolkit-object-tracking-2d
opendr-toolkit-object-detection-3d
opendr-toolkit-object-tracking-3d
opendr-toolkit-ambiguity-measure
```

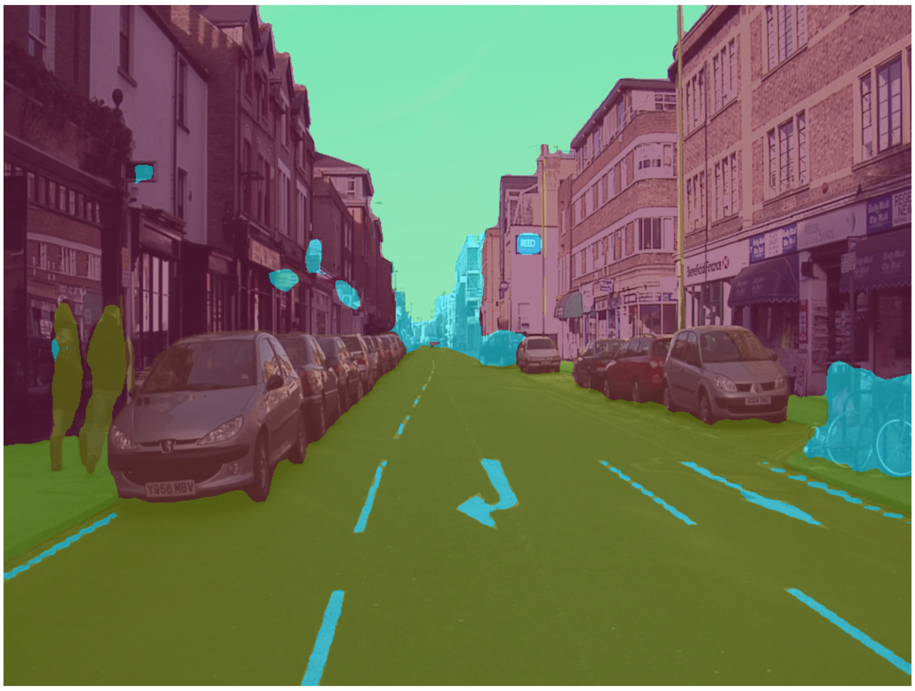


Figure 2: Result of running the semantic segmentation demo

4.2.4 Demo

After following the procedure mentioned above the virtual environment should be already active. If it is not the case, you can activate it with:

```
source venv/bin/activate
```

Now, for example, let's retrieve the *semantic segmentation* demo from OpenDR repository:

```
wget https://raw.githubusercontent.com/opendr-eu/opendr/master/projects/
↪  python/perception/semantic_segmentation/bisenet/inference_demo.py
```

When running `python3 inference_demo.py`, the model and test image will be downloaded and the result of the segmentation should be similar to what shown in Figure 2.

4.3 Installation using *docker*

4.3.1 Procedure

Appropriate dockerfiles that can run on any Linux system have been prepared and docker images are publicly available on dockerhub. First, docker needs to be installed in your system. For Ubuntu you can follow this procedure. When installed, running the OpenDR docker image is very easy. For example, for the CPU image all you need is to execute:

```
sudo docker run -p 8888:8888 opendr/opendr-toolkit:cpu_v2.0.0
```

or for the cuda-enabled one:

```
sudo docker run --gpus all -p 8888:8888 opendr/opendr-toolkit:cuda_v2.0.0
```

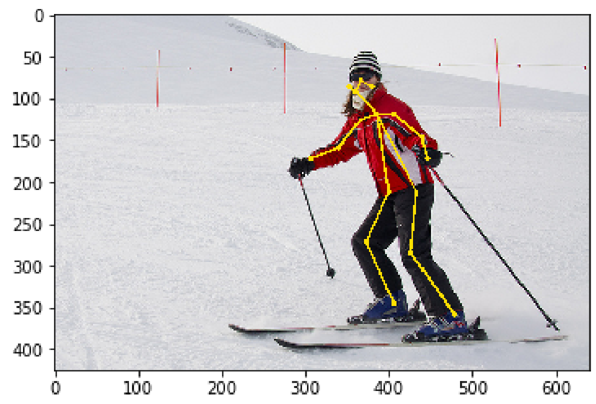


Figure 3: Result of running the pose estimation demo

Both commands will pull the image and launch it, and a *Jupyter* notebook server is started that listens on port 8888. This can be accessed by clicking on the link similar to `http://127.0.0.1:8888/?token=TOKEN` that appears in the console. Alternatively you can run an interactive session with:

```
sudo docker run -it opendr/opendr-toolkit:cpu_v2.0.0 /bin/bash
```

or

```
sudo docker run --gpus all -it opendr/opendr-toolkit:cuda_v2.0.0 /bin/bash
```

respectively for a cpu or cuda session. However, if you start an interactive session do not forget to enable the `venv` with the command:

```
source bin/activate.sh
```

If you want to display GTK-based applications from the Docker container (e.g., visualize results using OpenCV `imshow()`), then you should mount the X server socket inside the container:

```
xhost +local:root
sudo docker run -it -v /tmp/.X11-unix:/tmp/.X11-unix \
   -e DISPLAY=unix$DISPLAY opendr/opendr-toolkit:cpu_v2.0.0 /bin/bash
```

4.3.2 Demo

For example, in order to run the *pose estimation demo* you must first pull the latest image from `dockerhub` and run it:

```
sudo docker run -p 8888:8888 opendr/opendr-toolkit:cpu_v2.0.0
```

Now open the session in your browser, by clicking the link that appeared in the console, it should be similar to `http://127.0.0.1:8888/?token=TOKEN`.

Navigate to `projects/python/perception/lightweight_open_pose/demos/` and select the notebook `inference_tutorial.ipynb`. If you are using a CPU based docker, change the device to `cpu` in the second cell and execute the steps one by one. What you should expect at the end is shown in Figure 3.

To stop the Jupyter session you need to manually quit it.

5 ROS

With the second version of the toolkit, ROS and ROS2 nodes to interact with the toolkit have been provided for all included tools.

5.1 Environment setup for ROS

The instructions provided here will assume ROS `noetic` is already installed in your system, that a webcam is available and that you already installed the OpenDR toolkit (see section 4.1.1).

1. Move to the workspace: `cd projects/opendr_ws`
2. Install the package for the webcam: `sudo apt install ros-noetic-usb-cam`
3. Source the ROS installation: `source /opt/ros/noetic/setup.bash`
4. Build the workspace: `catkin_make`
5. Source it: `source devel/setup.bash`
6. Start roscore: `roscore &`

5.2 Demo

After activating the `virtualenv` with `source bin/activate.sh` at the root of the OpenDR folder, in one terminal you can launch the node responsible for publishing images with

```
rosrun usb_cam usb_cam_node
```

And in another terminal the `retinaface` face detection node can be started with

```
rosrun opendr_perception face_detection_retinaface.py
```

The annotated image stream is published in the topic `/opendr/image_boxes_annotated`, and the bounding boxes under `/opendr/faces`. Running `rqt_image_view` and selecting the `/opendr/image_boxes_annotated` topic allows to visualize the result.

6 ROS2

With the second version of the toolkit, ROS2 nodes to interact with the toolkit have been provided for all included tools.

6.1 Environment setup for ROS2

The instructions provided here will assume ROS2 `foxy` is already installed in your system, that a webcam is available and that you already installed the OpenDR toolkit (see section 4.1.1).

1. Activate the `virtualenv` at the root of the OpenDR folder: `source bin/activate.sh`
2. Move to the ROS2 workspace: `cd projects/opendr_ws_2`

3. Source the ROS2 installation: `source /opt/ros/foxy/setup.bash`
4. Build the workspace: `colcon build`
5. Source it: `source install/setup.bash`

6.2 Demo

From the ROS2 workspace folder available at `projects/opendr_ws_2`, and assuming the workspace was sourced with `source install/setup.bash`, you can launch the node responsible for publishing images with

```
ros2 run usb_cam usb_cam_node_exe
```

In order to see the camera images published by the webcam, in a new terminal you can run

```
ros2 run rqt_image_view rqt_image_view
```

and selecting the appropriate topic. In order to start the pose estimation node you can run in a different terminal

```
ros2 run opendr_perception pose_estimation
```

By running this command, the node will start publishing the pose messages it detects in the camera feed. You can see these messages being published by running, in a new terminal, the following command

```
ros2 topic echo opendr/poses
```

7 Using the OpenDR toolkit

OpenDR provides extensive documentation for each tool that is part of the toolkit and is available here, as well as pointing to the available demo. A list of the available ROS nodes is available here, specific instructions are provided for each of the available nodes. Excerpts from the documentation are available in the annex.

It is worth noting that the toolkit recorded an impressive 1100 unique GitHub clones (originated from users who wanted to have full access to all of the capabilities provided by the toolkit) in just one year. Indeed, the total number of clones exceeded 45000, which includes CI clones. Furthermore, an estimated 1700 pulls have been performed for the ready to use docker images (which mainly targets less experienced developers who want to directly try the toolkit) and an estimated 5000 pip downloads (originated from users who use selected parts of the Python API of the toolkit, e.g., individual tools without installing the whole toolkit). The larger number of pip downloads indicates that the target group is familiar with Python and prefers to install only the parts of the toolkit that are relevant to their needs. Note that the statistics for docker and pip have been adjusted to remove traffic that might have been automatically generated, e.g. by the CI system. Based on these statistics, OpenDR consortium estimates that at least 7500 downloads from developers have been performed in this period, exceeding by far the corresponding M48 KPI target (a total of 500 downloads for the toolkit).

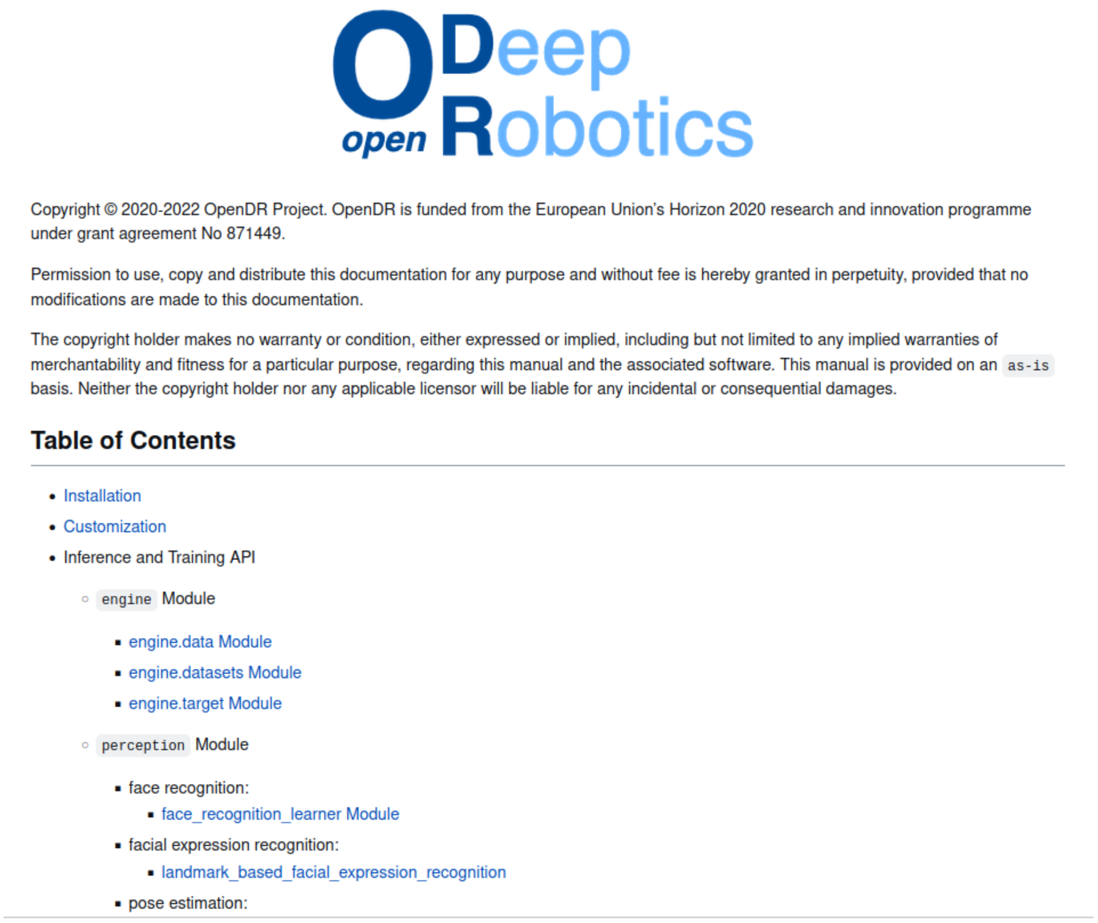
8 Customization

OpenDR can be readily customized to meet the needs of several application areas since the source code for all the developed tools is provided and the Apache 2.0 license is very permissive. Several ready-to-use examples, which are expected to cover a wide range of different needs, are provided. For example, users can readily use the existing ROS nodes by, for instance, including the required triggers or by combining several nodes into one to build more complex custom systems. An example of this is for instance the face recognition ROS node. In short, the user can use these nodes as a template to customize the toolkit to their needs. More in depth instructions on how to customize the toolkit are available in the dedicated page.

9 Conclusions

This document presents the work performed in WP7 about toolkit integration resulting in a second public version of the OpenDR toolkit. With this version several new tools have been added, ROS/ROS2 interfaces have been created for all available tools, user experience has been improved both by providing better documentation and by homogenizing the interface and how to interact with the tools, especially in ROS/ROS2.

10 Annex



Copyright © 2020-2022 OpenDR Project. OpenDR is funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449.

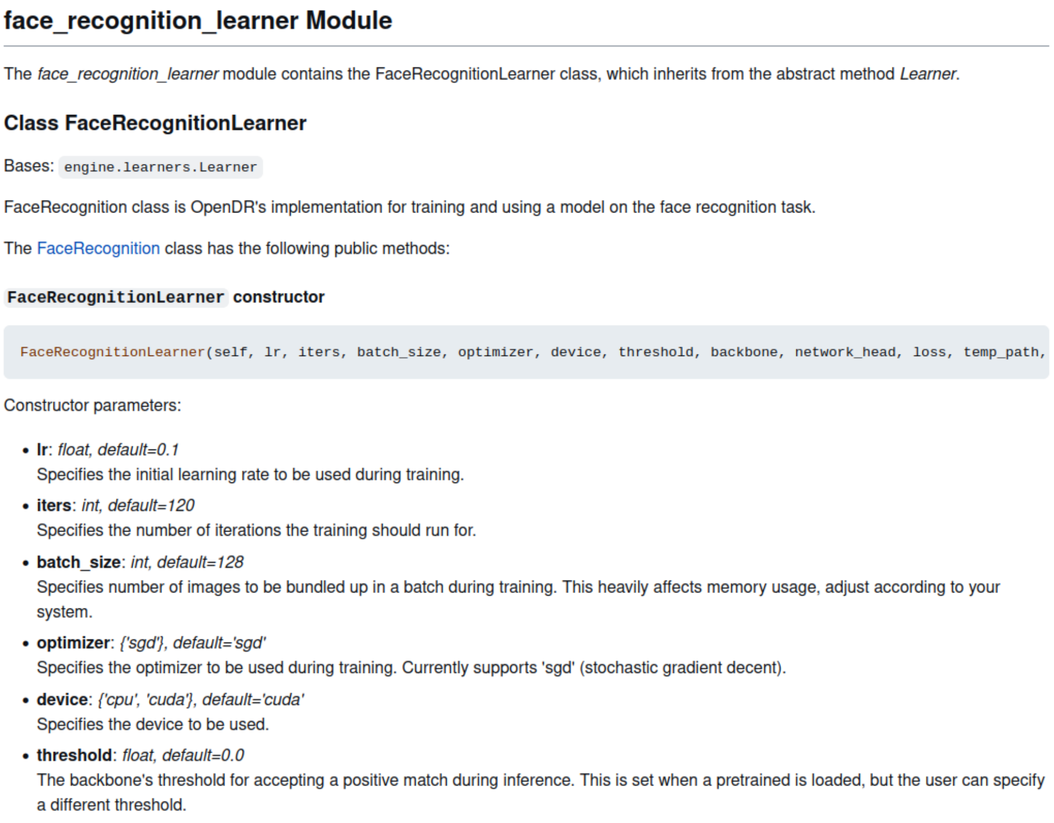
Permission to use, copy and distribute this documentation for any purpose and without fee is hereby granted in perpetuity, provided that no modifications are made to this documentation.

The copyright holder makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding this manual and the associated software. This manual is provided on an `as-is` basis. Neither the copyright holder nor any applicable licensor will be liable for any incidental or consequential damages.

Table of Contents

- Installation
- Customization
- Inference and Training API
  - `engine` Module
    - engine.data Module
    - engine.datasets Module
    - engine.target Module
  - `perception` Module
    - face recognition:
      - face_recognition_learner Module
    - facial expression recognition:
      - landmark_based_facial_expression_recognition
    - pose estimation:

Figure 4: Documentation index page.



face_recognition_learner Module

The *face_recognition_learner* module contains the FaceRecognitionLearner class, which inherits from the abstract method *Learner*.

Class FaceRecognitionLearner

Bases: `engine.learners.Learner`

FaceRecognition class is OpenDR's implementation for training and using a model on the face recognition task.

The FaceRecognition class has the following public methods:

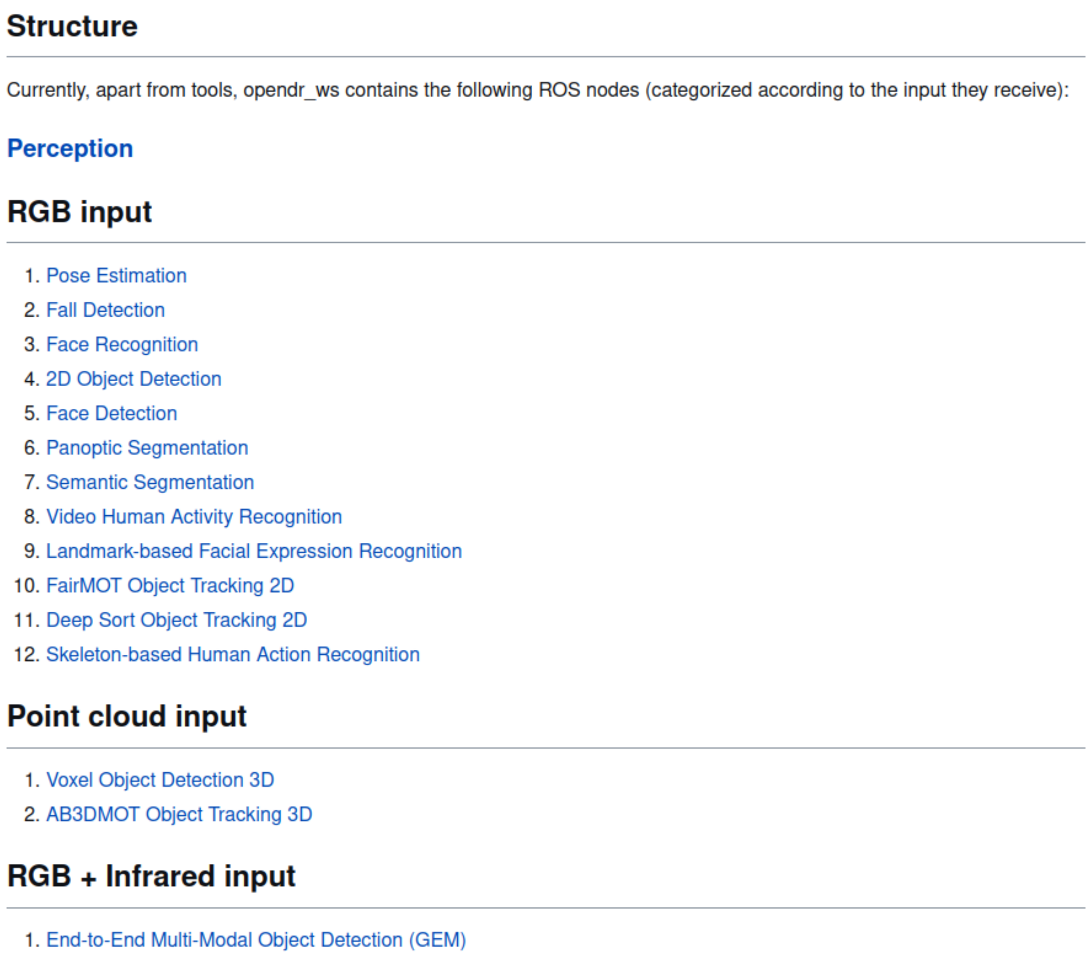
`FaceRecognitionLearner` constructor

```
FaceRecognitionLearner(self, lr, iters, batch_size, optimizer, device, threshold, backbone, network_head, loss, temp_path,
```

Constructor parameters:

- **lr**: *float, default=0.1*
  Specifies the initial learning rate to be used during training.
- **iters**: *int, default=120*
  Specifies the number of iterations the training should run for.
- **batch_size**: *int, default=128*
  Specifies number of images to be bundled up in a batch during training. This heavily affects memory usage, adjust according to your system.
- **optimizer**: *{'sgd'}, default='sgd'*
  Specifies the optimizer to be used during training. Currently supports 'sgd' (stochastic gradient decent).
- **device**: *{'cpu', 'cuda'}, default='cuda'*
  Specifies the device to be used.
- **threshold**: *float, default=0.0*
  The backbone's threshold for accepting a positive match during inference. This is set when a pretrained is loaded, but the user can specify a different threshold.

Figure 5: Excerpt from OpenDR documentation



Structure

Currently, apart from tools, opendr_ws contains the following ROS nodes (categorized according to the input they receive):

Perception

RGB input

1. Pose Estimation
2. Fall Detection
3. Face Recognition
4. 2D Object Detection
5. Face Detection
6. Panoptic Segmentation
7. Semantic Segmentation
8. Video Human Activity Recognition
9. Landmark-based Facial Expression Recognition
10. FairMOT Object Tracking 2D
11. Deep Sort Object Tracking 2D
12. Skeleton-based Human Action Recognition

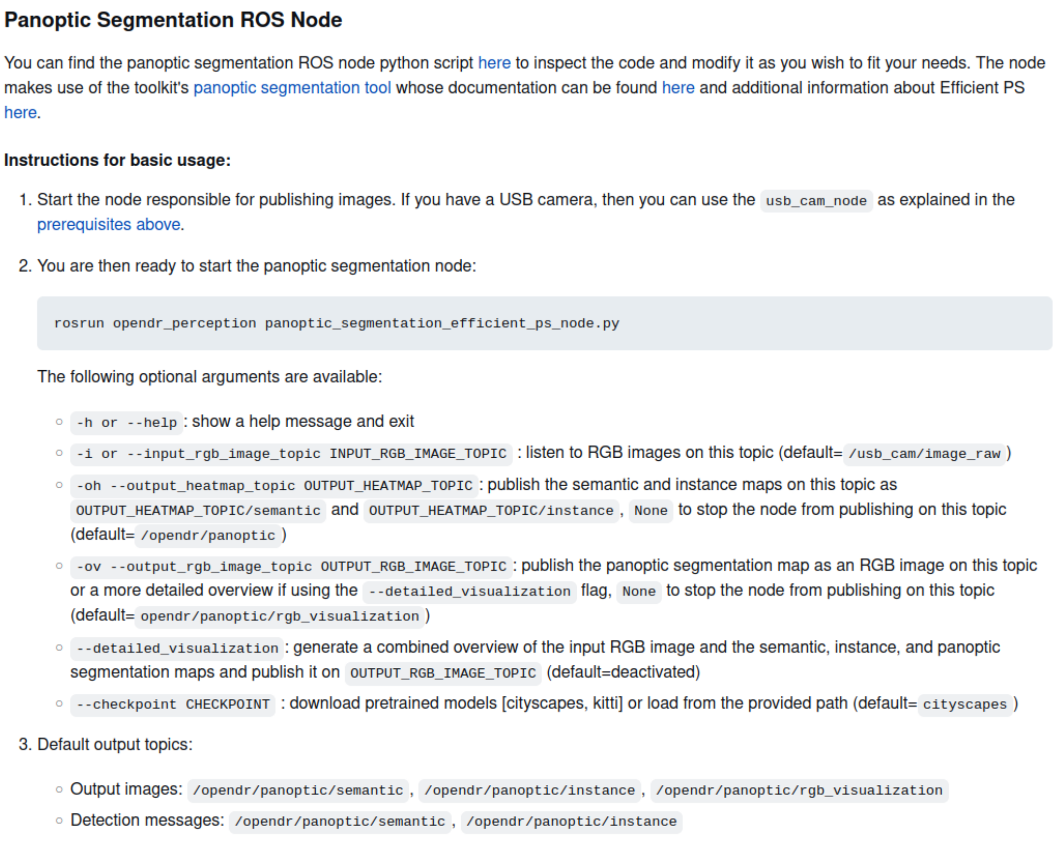
Point cloud input

1. Voxel Object Detection 3D
2. AB3DMOT Object Tracking 3D

RGB + Infrared input

1. End-to-End Multi-Modal Object Detection (GEM)

Figure 6: Index of ROS instructions for each tool



Panoptic Segmentation ROS Node

You can find the panoptic segmentation ROS node python script here to inspect the code and modify it as you wish to fit your needs. The node makes use of the toolkit's panoptic segmentation tool whose documentation can be found here and additional information about Efficient PS here.

Instructions for basic usage:

1. Start the node responsible for publishing images. If you have a USB camera, then you can use the `usb_cam_node` as explained in the prerequisites above.
2. You are then ready to start the panoptic segmentation node:

```
rosrun opendr_perception panoptic_segmentation_efficient_ps_node.py
```

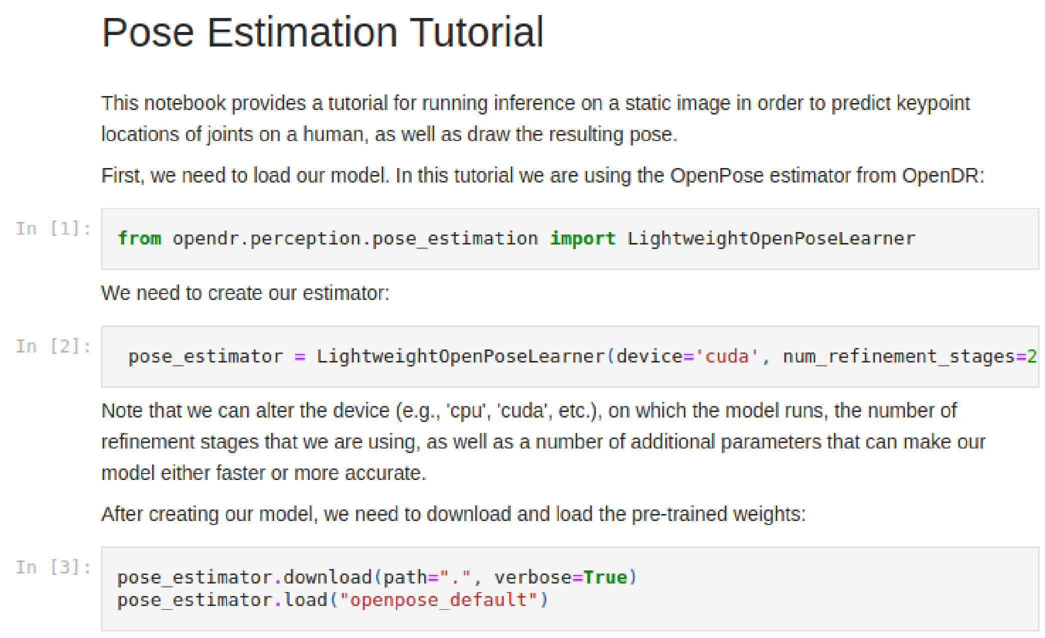
   The following optional arguments are available:

   - `-h or --help` : show a help message and exit
   - `-i or --input_rgb_image_topic INPUT_RGB_IMAGE_TOPIC` : listen to RGB images on this topic (default= `/usb_cam/image_raw`)
   - `-oh --output_heatmap_topic OUTPUT_HEATMAP_TOPIC` : publish the semantic and instance maps on this topic as `OUTPUT_HEATMAP_TOPIC/semantic` and `OUTPUT_HEATMAP_TOPIC/instance` , `None` to stop the node from publishing on this topic (default= `/opendr/panoptic`)
   - `-ov --output_rgb_image_topic OUTPUT_RGB_IMAGE_TOPIC` : publish the panoptic segmentation map as an RGB image on this topic or a more detailed overview if using the `--detailed_visualization` flag, `None` to stop the node from publishing on this topic (default= `opendr/panoptic/rgb_visualization`)
   - `--detailed_visualization` : generate a combined overview of the input RGB image and the semantic, instance, and panoptic segmentation maps and publish it on `OUTPUT_RGB_IMAGE_TOPIC` (default=deactivated)
   - `--checkpoint CHECKPOINT` : download pretrained models [cityscapes, kitti] or load from the provided path (default= `cityscapes`)

3. Default output topics:

   - Output images: `/opendr/panoptic/semantic` , `/opendr/panoptic/instance` , `/opendr/panoptic/rgb_visualization`
   - Detection messages: `/opendr/panoptic/semantic` , `/opendr/panoptic/instance`

Figure 7: ROS instructions of a tool



Pose Estimation Tutorial

This notebook provides a tutorial for running inference on a static image in order to predict keypoint locations of joints on a human, as well as draw the resulting pose.

First, we need to load our model. In this tutorial we are using the OpenPose estimator from OpenDR:

In [1]:
```
from opendr.perception.pose_estimation import LightweightOpenPoseLearner
```

We need to create our estimator:

In [2]:
```
 pose_estimator = LightweightOpenPoseLearner(device='cuda', num_refinement_stages=2
```

Note that we can alter the device (e.g., 'cpu', 'cuda', etc.), on which the model runs, the number of refinement stages that we are using, as well as a number of additional parameters that can make our model either faster or more accurate.

After creating our model, we need to download and load the pre-trained weights:

In [3]:
```
pose_estimator.download(path=".", verbose=True)
pose_estimator.load("openpose_default")
```

Figure 8: A Jupyter notebook prepared to showcase the usage of an OpenDR tool