

# Deep *CaT*-z:

Classification and Tracking in depth (z) using deep learning techniques

Software for real-time behavior recognition and automated closed-loop control of behavioral experiments, using depth videos

Ana Gerós, Paulo Aguiar

Faculty of Engineering of University of Porto (FEUP)  
Neuroengineering and Computational Neuroscience (NCN) Group,  
*i3S – Instituto de Investigação e Inovação em Saúde*  
Porto, Portugal  
[anaferos@ineb.up.pt]

**2022**

## Table of Contents

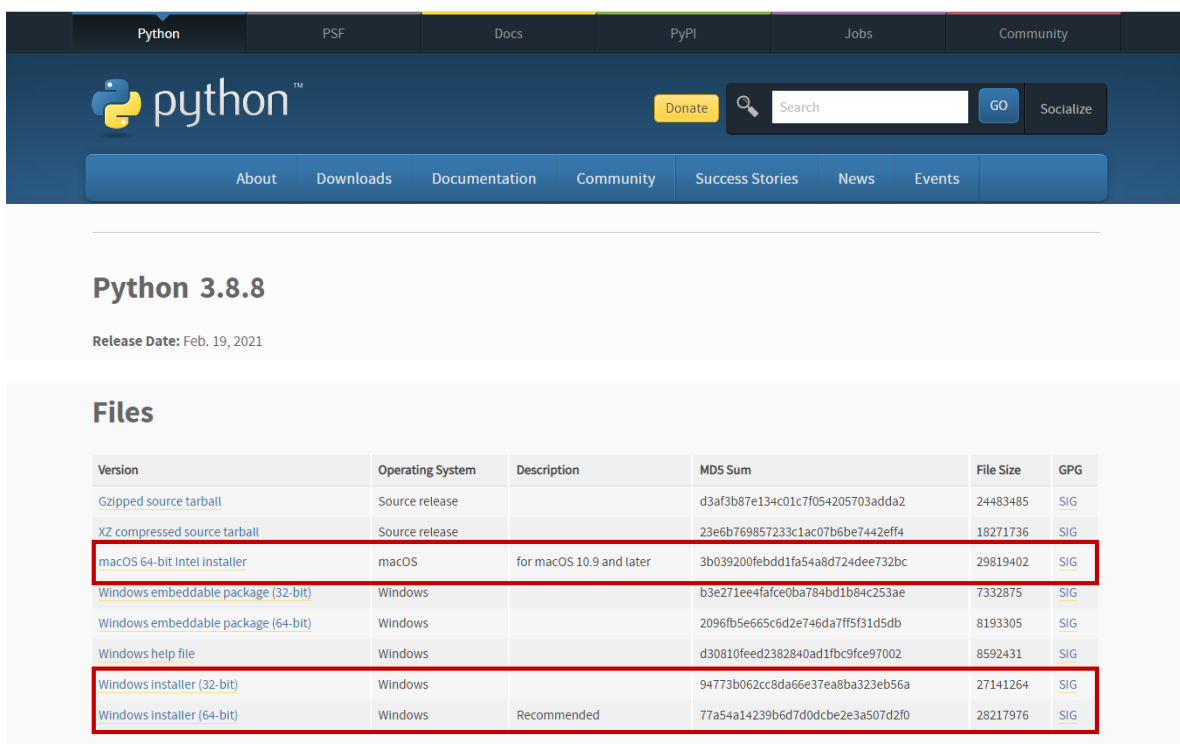
Installation .....	3
List of equipment/material/hardware .....	5
<i>DeepCaT-z</i> package.....	5
<b>1) Graphical user interface for the control of behavioral experiments: DeepCaT-z tool .....</b>	<b>6</b>
Step-by-step guide.....	8
Setup menu.....	8
Behavioral Experiments menu .....	14
<b>2) Configuration file – default parameters .....</b>	<b>17</b>
<b>3) Arduino control code .....</b>	<b>18</b>
<b>4) Preparing data for pre-training the deep learning models .....</b>	<b>21</b>
<b>5) Training the deep learning models – Behavior recognition and Segmentation for animal's detection .....</b>	<b>24</b>
Contact Us.....	26

# Installation

To install the *DeepCaT-z* tool, first, you need to have installed:

Python == 3.8.8

To install Python, go to <https://www.python.org/downloads/release/python-388/> and download the 3.8.8 Python version, according to your operating system and system type (64- or 32-Operating system).



The screenshot shows the Python 3.8.8 download page. The header includes the Python logo, navigation links (About, Downloads, Documentation, Community, Success Stories, News, Events), and a search bar. The main content area is titled "Python 3.8.8" with a release date of Feb. 19, 2021. Below this is a table of files for download.

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		d3af3b87e134c01c7f054205703adda2	24483485	SIG
XZ compressed source tarball	Source release		23e6b769857233c1ac07b6be7442eff4	18271736	SIG
<a href="#">macOS 64-bit Intel installer</a>	macOS	for macOS 10.9 and later	3b039200febdd1fa54a8d724dee732bc	29819402	<a href="#">SIG</a>
<a href="#">Windows embeddable package (32-bit)</a>	Windows		b3e271ee4fafce0ba784bd1b84c253ae	7332875	<a href="#">SIG</a>
<a href="#">Windows embeddable package (64-bit)</a>	Windows		2096fb5e665c6d2e746da7ff5f31d5db	8193305	<a href="#">SIG</a>
Windows help file	Windows		d30810feed2382840ad1fbc9fce97002	8592431	SIG
<a href="#">Windows installer (32-bit)</a>	Windows		94773b062cc8da66e37ea8ba323eb56a	27141264	<a href="#">SIG</a>
<a href="#">Windows installer (64-bit)</a>	Windows	Recommended	77a54a14239b6d7d0dcbe2e3a507d2f0	28217976	<a href="#">SIG</a>

During installation, check “Add Python 3.8 to PATH” and click “Install Now”:



OBS: If the installation was successful, the *Python* installation directory should be in the *PATH* of the *Environment Variables*. To confirm, in the *Start* menu, type "environment variables", press *Enter*, and then click on "Environment Variables..." in the lower right corner. In the *Path* field on the *User Variables for [User]* section, click on *Edit*, and check if the *Python38* installation directory is available. If not, insert a new line with the correct directory of *Python38* installation, which should be of the form of:

"C:\Users\[*User name*]\AppData\Local\Programs\Python\Python38".

Before going to the next step, make sure the "*Deep-CaT-z-Software-main*" folder is in an accessible directory.

So, once *Python* is installed, all the packages required to run the *DeepCaT\_z* tool need to be properly installed:

#### **For Windows:**

Click on the *InstallRequirements\_DeepCaT\_z.bat* file (in the "*Deep-CaT-z-Software-main*" folder) and wait until the installation is completed (this may take a few minutes).

### **For MAC:**

To open the *Command Line*, either open your *Applications* folder, then open *Utilities* and double-click on *Terminal*, or press *Command - spacebar* to launch *Spotlight* and type "*Terminal*," then double-click the search result. You'll see a small window with a white background open on your desktop.

After opening the *Command Line*, copy and paste the following lines, with *Enter* between them:

- 1) [insert the path of the *Deep-CaT-z-Software-main* folder]\get-pip.py
- 2) pip install -r [insert the path of the *Deep-CaT-z-Software-main* folder]\requirements\_DeepCaT\_z.txt

## List of equipment/material/hardware

To take full advantage of the software's capabilities, the following equipment is required:

- an [Intel® RealSense™ Depth Camera](#), connected via a *USB 3.0* port (to ensure maximum performance).

Camera drivers will be installed automatically once you connect the camera to your pc.

- (OPTIONAL) an Arduino microcontroller ([Mega 2560](#)), connected to a computer and to other input or output hardware modules (e.g., LEDs, levers, feeders, buttons).

To take advantage of the Arduino micro controller (or to edit the Arduino code and change the features created by default), it is necessary to install the *Arduino IDE*. To do this, download the executable in <https://www.arduino.cc/en/software> and follow the steps for a conventional installation (OBS: during the installation, different drivers need to be installed for the success of the operation).

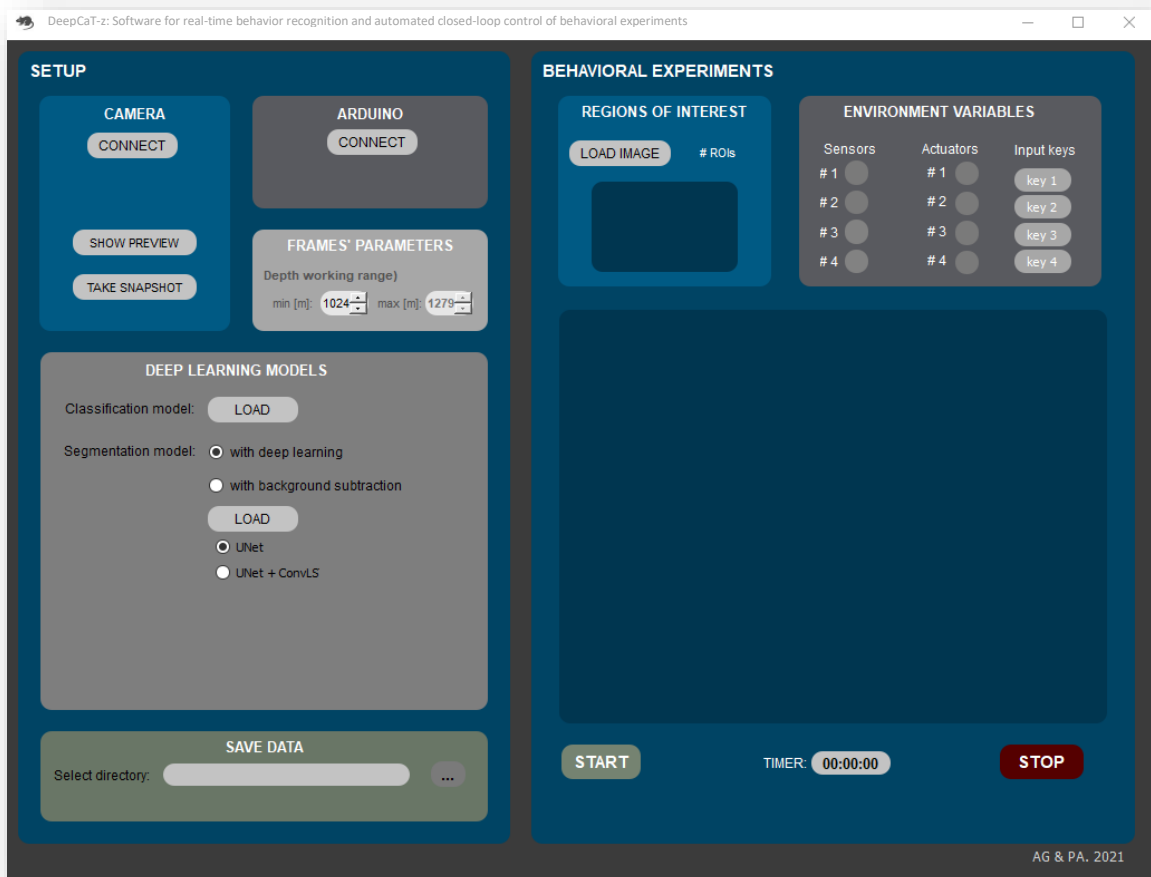
## DeepCaT-z package

The *DeepCaT-z* tool is a computational solution for real-time behavior recognition and automated closed-loop control of behavioral experiments, using depth videos. For the real-time automatic recognition of rodent's behavior and tracking, pre-trained deep learning models are applied for improved computational and task performances.

Besides the graphical user interface for real-time analysis of behavioral experiments, this tool contains additional functions for preparing the data for the deep learning models and for the training itself.

A step-by-step guide for each of these functionalities will be described below, and examples of pre-trained models, frames, and annotations for training/testing are also made available.

### 1) *Graphical user interface for the control of behavioral experiments: DeepCaT-z tool*



**Figure 1. *DeepCaT-z*:** Real-time behavior recognition and automated closed-loop control of behavioral experiments – main window.

To start the *DeepCaT-z* application, first make sure the camera and the Arduino (if needed) are properly connected to the computer.

Then, follow the steps to open the application:

**For Windows:**

Click on the *openDeepCaT\_z.bat* file (inside the “src” folder in the “*Deep-CaT-z-Software-main*” main folder) and wait until the application appears on the screen.

**For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following line:

```
>> [insert the path of the Deep-CaT-z-Software-main folder]\src\main_DeepCaT_Z.py
```

OBS: If during the execution of the application, the error (shown below) appears, close the application, and run the *Error\_Serial\_DeepCaT\_z.bat* file to solve the problem:

**For Windows:**

Click on the *Error\_Serial\_DeepCaT\_z.bat* file (inside the “*Deep-CaT-z-Software-main*” main folder).

**For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following lines, with *Enter* between them:

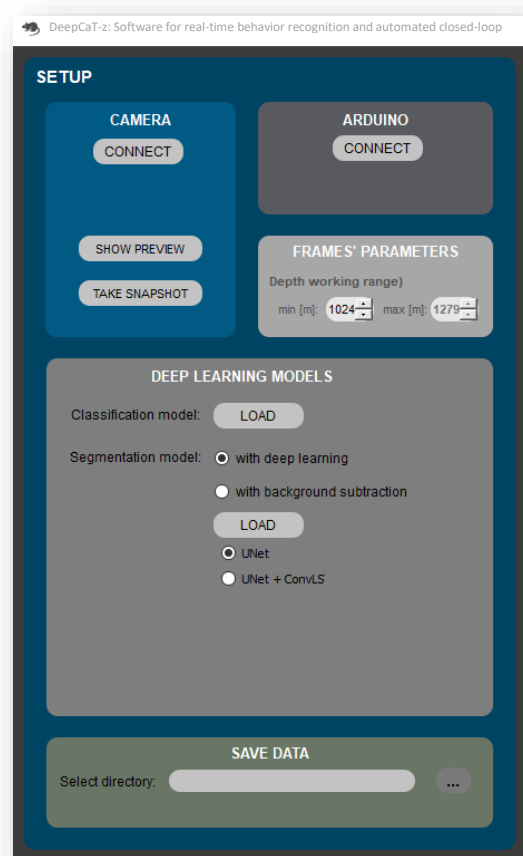
- 1) pip uninstall serial
- 2) pip uninstall pyserial
- 3) pip install pyserial

**ERROR:**

```
QWindowsContext: OleInitialize() failed: "COM error 0xffffffff80010106 RPC_E_CHANGED_MODE
(Unknown error 0x080010106)"
Traceback (most recent call last):
  File "main_DeepCaT_Z.py", line 2477, in <module>
    ui.setupUi(MyMain)
  File "main_DeepCaT_Z.py", line 1067, in setupUi
    self.serial_port = serial.Serial()
AttributeError: module 'serial' has no attribute 'Serial'
```

## Step-by-step guide

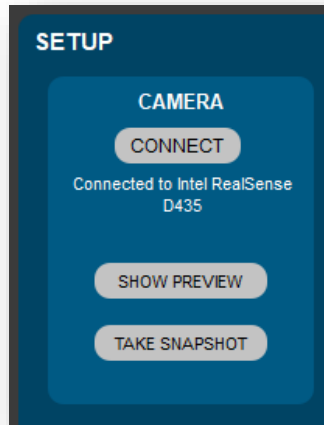
### Setup menu



**Figure 2. Setup menu:** it provides options for camera and Arduino connections, depth frames' and deep learning models' parameters, and finally selection of the directory to save experimental data.

1. In the GUI, click the *Connect* button for camera connection, and if the camera was correctly identified and connected to the computer, a message will appear, as follows:





Otherwise, the *"No Intel Device connected!"* message will appear.

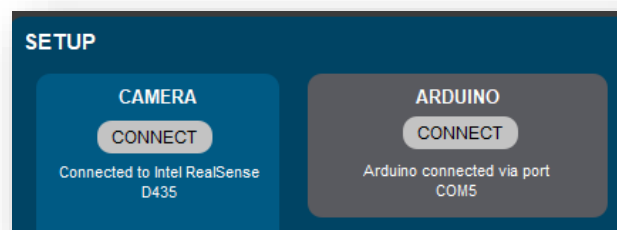
2. To preview the camera's field of view, click on *Preview*. A new window will appear with the depth live stream.

OBS: This live stream preview already shows the depth working range pre-defined in the *Frames' Parameters* section (on the right). Further explanation will be given in step 5.

3. To take a snapshot of the camera's field of view, click on *Take Snapshot*. A new window will appear where it is possible to define the directory to where the snapshots will be saved and finally one RGB frame and one depth frame (an RGB-D registered pair of frames) will be automatically saved.

OBS: These snapshots may be useful to create user-defined regions of interest (see *Behavioral Experiments menu* guide, step 1). It is important to point out that the RGB frames are registered to the depth frames and, for that reason, the visible color area is smaller (with the presence of a black frame). This black zone does not mean that this area will be discarded from the recorded depth frames. It just means that there is no color information for this particular area. Thus, it is advisable to use depth frames to generate regions of interest whenever possible.

4. Click the *Connect* button for Arduino connection and, again, and if the microcontroller was correctly identified and connected to the computer, a message will appear, as follows:



Otherwise, the *"Problems connecting to Arduino!"* message will appear.

5. The *Frames' Parameters* section allows modification of the depth working range. The acquired depth frames are 16 bits. However, the frames used for automatic classification/tracking with deep neural networks are 8 bits. For that reason, it is necessary to choose a relevant depth working range that contains the 255 possible values for the 8-bits scenario, as a way to avoid losing relevant information. To choose this working range, it is necessary to take into account the depth range of the object of interest and define the minimum depth value from which it is relevant to maintain (using the *min [m]* spin box object). The maximum value will be automatically calculated, adding up 255.
6. The *Deep Learning Models* section contains all the features for automatically classifying and tracking the animal. It should be noted that the classification and tracking are independent of each other, and do need to be performed simultaneously (e.g., animal tracking can be performed independently, without the behavior classification task).

- *Classification model:*

If the objective is to automatically classify animal behavior, it is necessary to use a model that has been previously trained. The steps to train a model will be given later, in section 5) *“Training the deep learning models – Behavior recognition and Segmentation for animal’s detection”*, together with the chosen architecture and all necessary parameters.

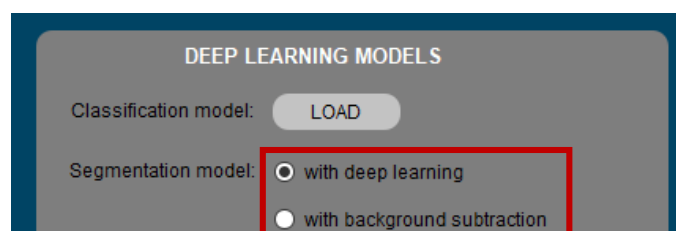
To load the model, click on *LOAD*, where a window will appear allowing you to browse the directories and choose the file that contains the trained model. The file must contain be of the following format: [NAME\_MODEL].pth.

OBS: A previously trained model is provided as an example for testing the application (file name: model\_behav\_rnn\_8fps\_128.pth). This model is prepared to receive frames with a pixel’s resolution of 128x128, and an input sequence of 11 depth frames.

- *Segmentation model:*

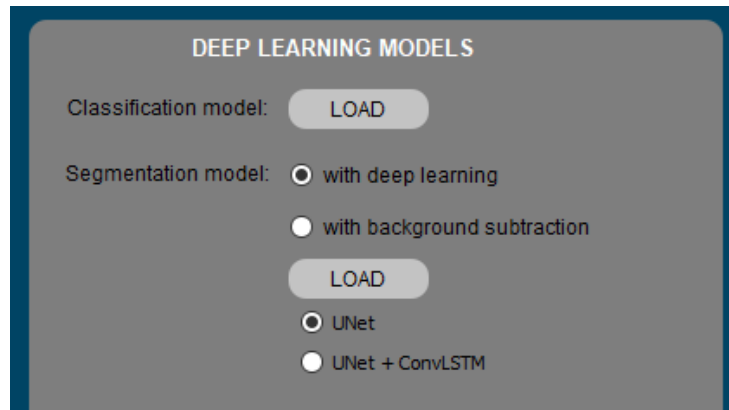
The segmentation step is the first to track the animal: the pixels that belong to the object of interest are separated from those of the background (segmentation task), to later calculate the coordinates of the centroid (center of mass) of the animal.

Two different methods can be used to perform the segmentation: deep learning models or a simple background subtraction model. The segmentation method can be chosen by clicking on one of the available radio buttons: *“with deep learning”* or *“with background subtraction”*:



Depending on the method chosen, different features will appear in the application:

- **with deep learning**: two different methods can be used to do segmentation using deep learning models: 1) model with U-Net as backbone architecture but with fewer parameters/layers; 2) model with U-Net and *ConvLSTM* layers for temporal integration. For more information regarding model architectures, see the Materials and Methods Section in the original publication or check *classificationModel\_functions.py* and *segmentationModel\_functions.py* files in the *DeepCaT\_z* directory.

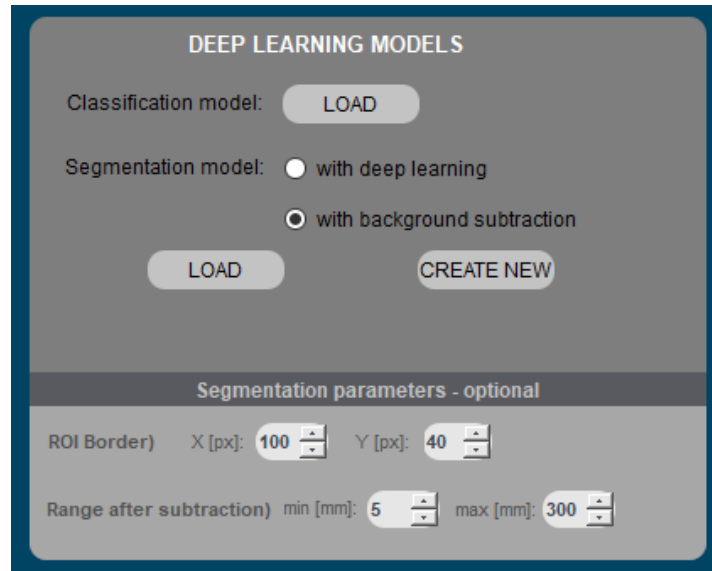


Click on one of the *radio buttons'* options and then click on **LOAD** to import the models. The file must contain be of the following format: [NAME\_MODEL].hdf5.

OBS: Two previously trained models are provided as examples for testing the application (file name: *model\_unet\_NOTIME\_8fps\_128.hdf5* or *model\_unet\_convlstmv3\_8fps\_128.hdf5*). These models are prepared to receive frames with a pixel's resolution of 128x128, and an input sequence of 11 depth frames.

- **with background subtraction**: the background subtraction method is a simpler method for segmenting objects of interest, which uses a background model (an image that is representative of the environment's background) from which each video frame is subtracted. The object of interest will be segmented by pixel-by-pixel subtraction of the background value.

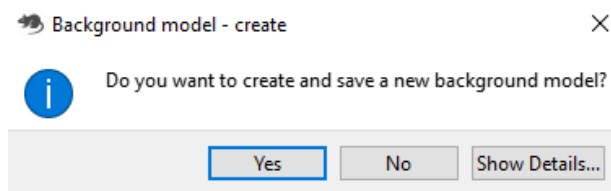
In this way, it is necessary to load a previously created background or create a new background model.



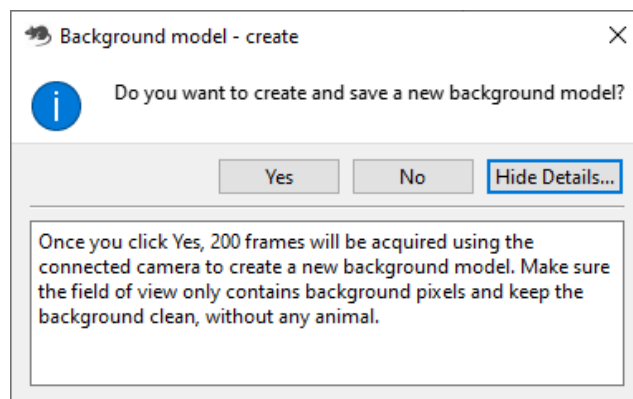
To load a background model, click on the **LOAD** button, and select a *.png* image containing background information. This image must be of the same resolution as the frames acquired by the depth camera to be used and must be 16-bits (depth information is always encoded in 16-bits).

OBS: A background model image is provided as an example for testing the application (file name: *modelBack\_200\_uint16.png*).

To create a new background model, click on the **CREATE NEW** button and a new window will appear to confirm the operation:



OBS: Click on **Show Details...** to check further information regarding this operation:



After clicking **Yes**, a new window will appear to show the created background model, which will be automatically saved to the application directory under the name: *modelBack\_new\_[DATE].png*.

The *Segmentation Parameters – optional* section contains the main parameters for the background subtraction method:

**Border – X [px] or Y [px]:** number of pixels that will be ignored (on each side of the image) to create an ROI margin (OBS: it can be useful to account for shadows or alterations in the walls of the behavioral setups).

**Range after subtraction: min [mm] or max [mm]:** range of depth values that are included in the analysis after background subtraction. Out of this range, the values are considered noise (below or above camera's precision).

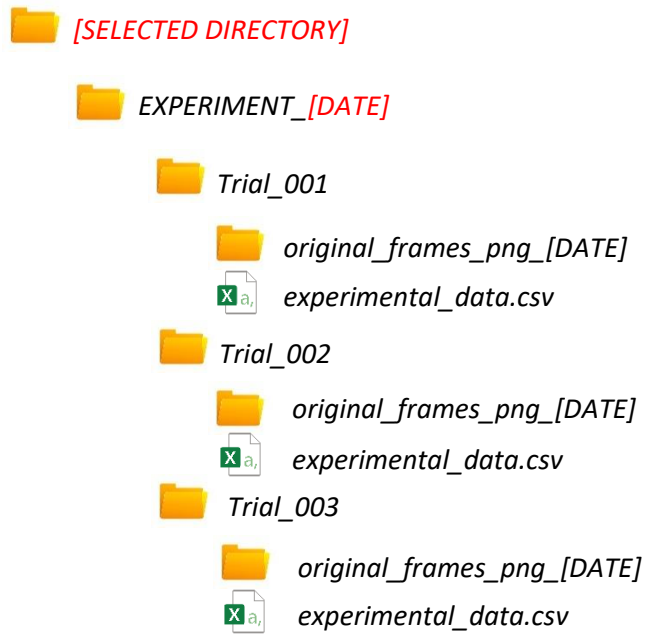
To see the effect these parameters have on the depth frame, click on *SHOW PREVIEW* (in the *CAMERA* section, on the upper left corner).

7. The *SAVE DATA* section allows choosing the directory where all the results of an experimental run will be saved, namely:

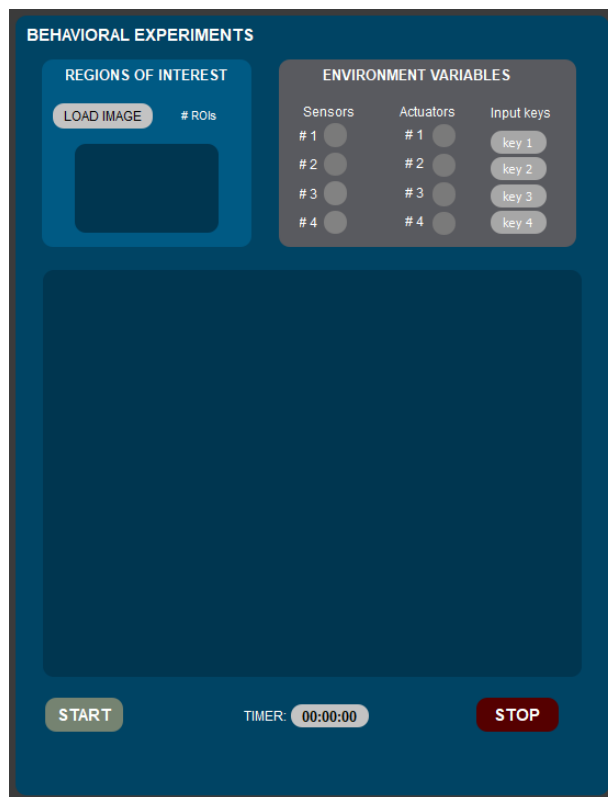
- preprocessed (8-bit) depth frames, in a folder under the name *original\_frames\_png\_[DATE]*;
- .csv file with detailed information of all frames that were acquired during the experiment, namely timesteps (in milliseconds), behavioral label, coordinates of animal's centroid (x, y, and z, in pixels), ROI identification, key-pressed identification, and the output of the Arduino (state of input/sensors and output/actuators).

The first *N* frames and corresponding information are not saved: the *N* initial frames are needed for the behavior/position predictions and from which it is not possible to extract information (OBS: the number of discarded frames is related to the size of the *STEPS\_LIST* variable in the *Configuration File*; for more information see section *Configuration file – default parameters*).

It is important to note that within the chosen directory, several folders will be created for each independent run (i.e., a different run is considered each time the user clicks on *START* without having selected a new directory). In this way, the configuration of the folders will be as follows:



[Behavioral Experiments menu](#)

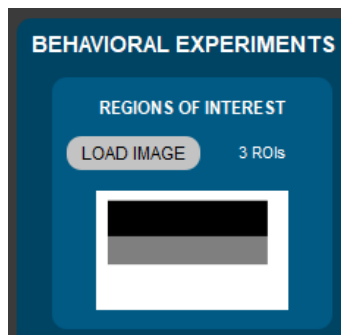


**Figure 3. Behavioral Experiments menu:** it provides features for the selection of user-defined regions of interest and visualization of the environment variables state and camera's field of view during the experiments.

1. In the *Regions of Interest* section, it is possible to load an image containing user-defined regions of interest, that will be used immediately after the tracking. Depending on the tracking position of the animal, the software identifies in which region the animal is and saves that information for further analysis. It is important to note that this step is not mandatory (it is an extra functionality of the software).

The loaded image must be of the same resolution as the original depth frames acquired by the camera (.png format) and must identify, with different colors, the different areas of interest (mROI).

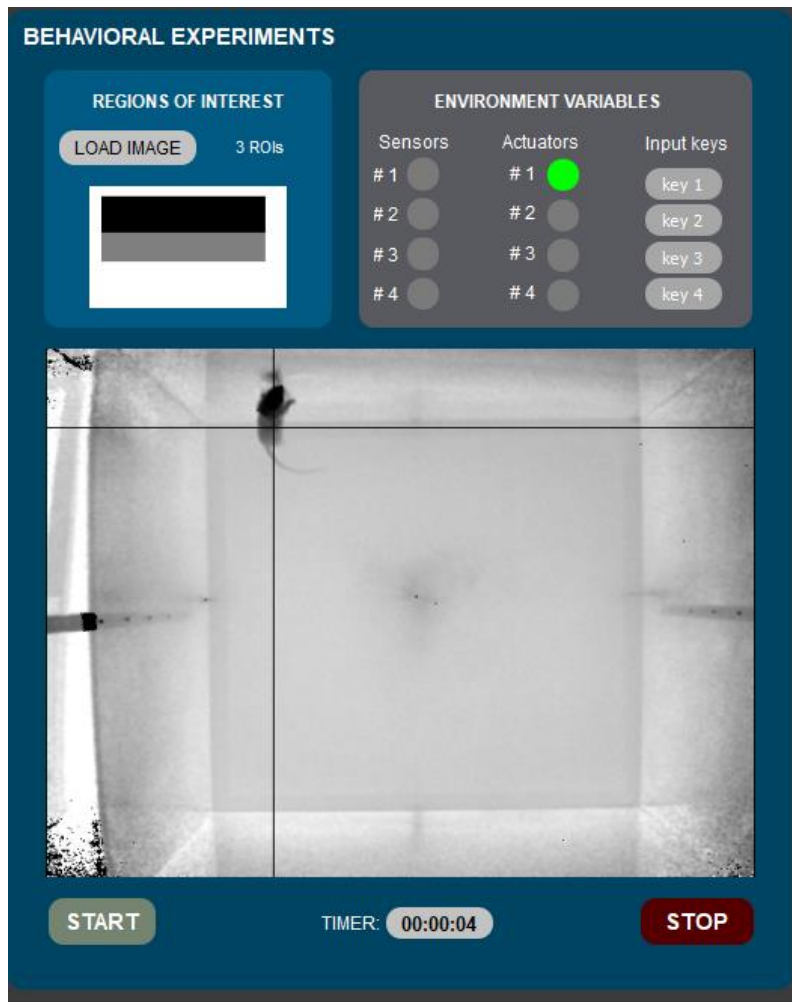
To create this mROI image, the snapshot functionality (*Take Snapshot* button in the *Camera* section) can be useful: the recorded images (depth or RGB) can be used as templates to help identify the regions of interest. To do this, open an image editing tool (e.g., *Paint* in Windows), and overlay regions that are of interest, identified by a unique color. A preview of the loaded image is shown, together with the identification of the number of regions available.



OBS: A mROI image is provided as an example for testing the application (file name: *mROI\_example.png*).

2. Once everything is ready (camera, Arduino, pre-trained models, parameters, and optionally mROIs), click *START*.

The acquisition of the depth frames will automatically start, and a preview of the camera's field of view is available in the central panel, together with the position of the animal's centroid, if available:



The *Environment Variables* section will show all input (sensors) and output (actuators) states (on – **green**; off - **gray**), and keyboard buttons' states. The response of these inputs and outputs is defined by the user, through an Arduino user-defined function. This function allows access to all critical variables for the control, such as sensors' and actuators' states, and animal's position and behavior. Inside this function, the user can easily define the conditions of stimuli-response that characterize each behavioral test experiment. To learn how to modify this function to achieve specific tasks for each experimental setup, check Section 3 – Arduino control code.

3. To stop the automatic acquisition and detection, click **STOP**, and all acquired frames and experimental data are automatically saved in the defined directory.
4. If it is necessary to do new recordings/detections, just click **START** again, and a new trial will automatically start (a new folder will be created in the previously defined directory).



## 2) Configuration file – default parameters

Along with the *main\_DeepCaT\_Z.py* file available in the *DeepCaT\_z* folder, there is a configuration file named "*config\_DeepCaT\_z*", which contains all the default parameters that support the application.

The parameters are as follows:

**ORIGINAL\_SIZE\_Y:** original resolution on the Y-axis (number of pixels of the width (rows)). This resolution is defined by the camera that is used to acquire the frames. By default, the original resolution was set to 480x640 to match *Intel Real Sense* cameras' resolution (best performance achieved with this pixel's resolution).

**ORIGINAL\_SIZE\_X:** original resolution on the X-axis (number of pixels of the length (columns)).

**IMG\_SIZE:** pixel resolution of resized depth frames. Deep learning models receive as input a sequence of frames of a different resolution than the original.

**CHANNELS\_SIZE:** number of channels of the frames. By default, the number of channels was set to 1 (grayscale).

**TIMEOUT\_ARDUINO:** maximum milliseconds to wait for serial data.

**BAUDRATE:** data rate in bits per second (baud) for serial data transmission.

**GPU\_AVAILABLE:** GPU for deep learning models' processing. If True, GPU will be used. If False, GPU will not be used. It is necessary to confirm that the computer where the acquisitions/processing is carried out has a GPU card.

**STEPS\_LIST:** a list containing the IDs of the depth frames to be included in the input sequence, in the form of: (ID of the last frame to be included, id of the current frame, step between frames). For more information regarding model architectures and input parameters, see the Materials and Methods Section in the original publication.

**NCLASSES:** number of behavioral events/classes for the behavior classification task. By default, the number of classes is 4: 0 – Standstill; 1 – Walking; 2 – Rearing; 3 – Grooming.

**KERNEL\_EROSION:** kernel matrix for erosion operation (post-processing step after background subtraction to clean isolated pixels on the mask).

**NUM\_FRAMES\_TO\_CREATE\_MODEL:** number of frames that are acquired to create the background model using the background subtraction method.

**MINRATRANGE/MAXRATRANGE:** default minimum and maximum depth frames in the spin boxes of the *Segmentation parameters – optional* section.

To edit the information in the configuration file, open any source code or *Python* editor (such as *Spyder* or *Notepad*), and edit its content.

### 3) *Arduino control code*

The Arduino file (.ino) made available (in the *ARDUINO\_CODE* folder) contains all the features to program the Arduino Mega in receiving input from 4 sensors and 4 actuators (maximum).

The communication protocol between the computer and the Arduino consists of a sequence of input characters (communication: computer -> Arduino) and output (communication Arduino -> computer) initially defined as follows:

- **Computer -> Arduino:**

Sequence: S,X,Y,Z,L,K

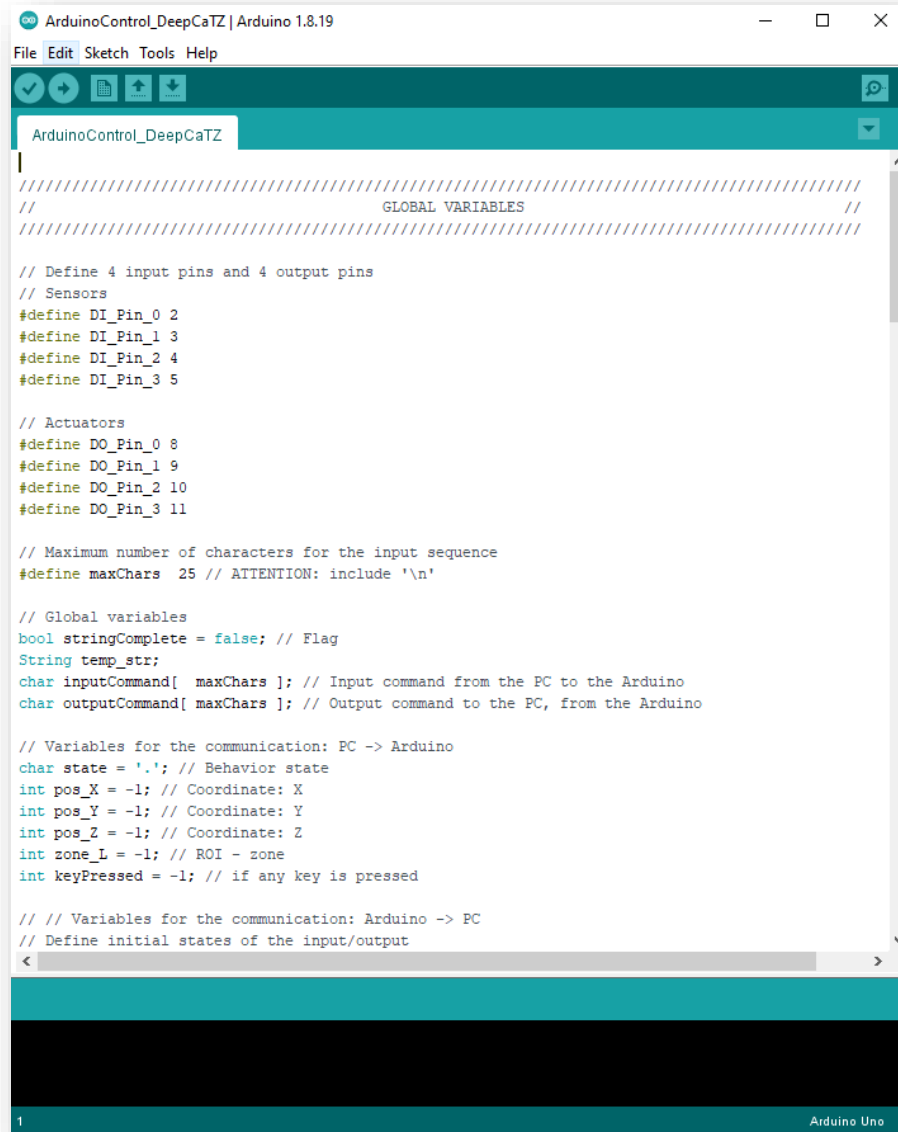
Behavioral state	S <char>	S: <i>standstill</i> W: <i>walking</i> R: <i>rearing</i> G: <i>grooming</i> N: none
Centroid Pos X <int>	X	
Pos Y <int>	Y	
Pos Z <int>	Z	
Region (mROI) <int>	L	
Keypress <int>	K	

- **Arduino -> Computer:**

Sequence: DI\_0,DI\_1,DI\_2,DI\_3,DO\_0,DO\_1,DO\_2,DO\_3

Input sensors DI <int>	on: 1 off: 0
Output actuators DO <int>	on: 1 off: 0

To edit the Arduino code, open the Arduino application (previously installed) and open the "*ArduinoControl\_DeepCaTZ.ino*" file (inside the *ArduinoControl\_DeepCaTZ* folder):



The screenshot displays the Arduino IDE interface with the following components:

- Title Bar:** ArduinoControl\_DeepCaTZ | Arduino 1.8.19
- Menu Bar:** File, Edit, Sketch, Tools, Help
- Toolbar:** Includes icons for opening, saving, and running the sketch.
- Tab Bar:** Shows the active sketch file: ArduinoControl\_DeepCaTZ.
- Code Editor:** Contains the following C++ code:

```
////////////////////////////////////  
//                                GLOBAL VARIABLES                                //  
////////////////////////////////////  
  
// Define 4 input pins and 4 output pins  
// Sensors  
#define DI_Pin_0 2  
#define DI_Pin_1 3  
#define DI_Pin_2 4  
#define DI_Pin_3 5  
  
// Actuators  
#define DO_Pin_0 8  
#define DO_Pin_1 9  
#define DO_Pin_2 10  
#define DO_Pin_3 11  
  
// Maximum number of characters for the input sequence  
#define maxChars 25 // ATTENTION: include '\n'  
  
// Global variables  
bool stringComplete = false; // Flag  
String temp_str;  
char inputCommand[ maxChars ]; // Input command from the PC to the Arduino  
char outputCommand[ maxChars ]; // Output command to the PC, from the Arduino  
  
// Variables for the communication: PC -> Arduino  
char state = '.'; // Behavior state  
int pos_X = -1; // Coordinate: X  
int pos_Y = -1; // Coordinate: Y  
int pos_Z = -1; // Coordinate: Z  
int zone_L = -1; // ROI - zone  
int keyPressed = -1; // if any key is pressed  
  
// // Variables for the communication: Arduino -> PC  
// Define initial states of the input/output
```
- Serial Monitor:** A black area at the bottom, currently empty.
- Status Bar:** Shows the board type as 'Arduino Uno'.

Figure 4. Arduino IDE interface with source code.

Initially, it is necessary to make the correct connections between the Arduino board and the external inputs/outputs. In this particular example, the Digital Input (DI) defined as zero (0) is connected to PIN 2, DI\_1 to PIN 3, etc., as well as the Digital Output (DO) defined as 0, is connected to PIN 8 in the Arduino board:

```
// Define 4 input pins and 4 output pins
// Sensors
#define DI_Pin_0 2
#define DI_Pin_1 3
#define DI_Pin_2 4
#define DI_Pin_3 5

// Actuators
#define DO_Pin_0 8
#define DO_Pin_1 9
#define DO_Pin_2 10
#define DO_Pin_3 11
```

All other global variables are necessary for the proper functioning of the program and allow defining the initial state input/output variables and the state of the sensors and actuators (ON or OFF).

To modify the conditions of stimuli-response that characterize each behavioral test experiment, it is necessary to change the *User\_Defined\_Control\_Fun()* function inside the **USER DEFINED FUNCTION** section:

```

ArduinoControl_DeepCaTZ
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                     USER DEFINED FUNCTION                                     //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Define all signals from input command to the output actuators.
void User_Defined_Control_Fun() {

    // EXAMPLE:
    // Everytime the input behavioral state is
    // - "Standstill" (S), the output actuator 0 is turned ON; if not: turn OFF
    // - "Walking" (W), the output actuator 1 is turned ON; if not: turn OFF
    // - "Rearing" (R), the output actuator 2 is turned ON; if not: turn OFF
    // - "Grooming" (G), the output actuator 3 is turned ON; if not: turn OFF

    // Actuator 0 - DO_0
    // Define if needed
    if( state == 'S' ) {
        DO_0 = HIGH;
    }
    else {
        DO_0 = LOW;
    }

    // Actuator 1 - DO_1
    if( state == 'W' ) {
        DO_1 = HIGH;
    }
    else {
        DO_1 = LOW;
    }

    // Actuator 2 - DO_2
    if( state == 'R' ) {
        DO_2 = HIGH;
    }
    else {
        DO_2 = LOW;
    }

    // Actuator 3 - DO_3
    // Define if needed
    if( state == 'G' ) {
        DO_3 = HIGH;
    }
}

```


Within this function, it is possible to find a pre-defined example where different actuators are activated depending on the behavioral state of the animal. Modify each one of the conditions to obtain different responses from the actuators that are connected to the Arduino.




#### 4) *Preparing data for pre-training the deep learning models*


The “DeepCaT\_z” main folder contains scripts (in the *TRAIN\_MODELS* folder) to prepare the data and train the deep learning models, as well as some example data for testing all functions ([file name: dataset\\_train100\\_val50\\_test50\\_128\\_example.zip](#)). To train the classification or segmentation models, it is necessary, along with the depth frames, the corresponding annotations/ground truth (classes/behaviors or segmentation masks), to construct the training and validation sets. The testing set is optional and can be used to check the performance of the trained model.




The original depth data must follow the following configuration:


 [ORIGINAL DATA\_FOLDER\_NAME]




 *train*

-  *frames*
-  *labels*
-  *masks*

 *val*

-  *frames*
-  *labels*
-  *masks*

 *test*

-  *frames*
-  *labels*
-  *masks*

The *frames* folder should contain a list of *.png* files with 16-bit depth frames. The *labels* folder should contain a list of *.txt* files with the class or behavior associated with each frame in the previous folder. The *masks* folder must contain a list of *.png* binary masks (with a “\_seg” termination name) of the animal segmentation (0 - background; 1 - animal).

For the acquisition of raw depth frames, we suggest using the *Data Acquisition* module of the *CaT-z* software ([https://github.com/CaT-zTools/CaT-z\\_Software](https://github.com/CaT-zTools/CaT-z_Software), under the modules *CaT\_z\_dataAcquisitionGUI\_RealSense* or *CaT\_z\_dataAcquisitionGUI\_Kinect*).

For the manual annotation of animal's behavior in each depth frame, we suggest using the Visualization and Annotation module of the *CaT-z* software ([https://github.com/CaT-zTools/CaT-z\\_Software](https://github.com/CaT-zTools/CaT-z_Software)).

For the annotation of the segmentation masks, we suggest using the *GrabCut* algorithm ([https://docs.opencv.org/3.4/d8/d83/tutorial\\_py\\_grabcut.html](https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html)), for background separation with minimal user interaction.

After collecting the frames and corresponding annotations, the next step is training the models:

- Prepare data for the training of the **segmentation** models

To improve the performance of the models and allow for more diversity of the training set, data augmentation is a common step. The *preprocess\_augSequences\_Segmentation.py* file, inside the SEGMENTATION folder, contains all functions needed for augmenting the data (depth frames and corresponding segmentation masks).

The original data needs to have the format as previously explained. To edit the default parameters for the augmentation operations, open any source code or *Python* editor (such as *Spyder* or *Notepad*), load the *config\_preprocess\_augSequences\_DeepCaT\_Z.py* file, and change it if necessary.

To run the *preprocess\_augSequences\_Segmentation.py* file (main file):

**For Windows:**

Click on the *preprocess\_Segmentation\_DeepCaT\_z.bat* file (inside the "TRAIN\_MODELS\SEGMENTATION" folder) and wait until the preprocessing is completed.

**For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following line:

```
>> [insert the path of the Deep-CaT-z-Software-main folder]\ TRAIN_MODELS\SEGMENTATION\ preprocess_augSequences_Segmentation.py
```

The augmented dataset will be saved in a new directory and can then be used to train the segmentation models.

- Prepare data for the training of the **classification** models

The only pre-processing that depth frames undergo before running training scripts is resizing to the standard pixel's resolution.

The *preprocess\_Classification.py* file, inside the CLASSIFICATION folder, contains all functions and parameters needed for resizing the data.

The original data needs to have the format as previously explained. To edit the default parameters for the resizing operations, open any source code or *Python* editor (such as *Spyder* or *Notepad*), load the *config\_preprocess\_Classification\_DeepCaT\_Z.py* file, and change it if necessary.

To run the *preprocess\_Classification.py* file (main file):

**For Windows:**

Click on the *preprocess\_Classification\_DeepCaT\_z.bat* file (inside the “TRAIN\_MODELS\CLASSIFICATION” folder) and wait until the preprocessing is completed.

**For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following line:

```
>> [insert the path of the Deep-CaT-z-Software-main folder]\TRAIN_MODELS\CLASSIFICATION\preprocess_Classification.py
```

## 5) *Training the deep learning models – Behavior recognition and Segmentation for animal’s detection*

The training of the segmentation and classification models are done separately, and two independent folders (SEGMENTATION and CLASSIFICATION folders) were organized with the necessary functions and main code for each task.

- Training the **segmentation** model

The *main\_segmentationTask\_UNet.py* file, inside the SEGMENTATION folder, contains all functions needed for training the U-Net and the extended U-Net with ConvLSTM networks. To edit the default parameters for the segmentation training operations, open any source code or *Python* editor (such as *Spyder* or *Notepad*), load the *config\_Segmentation\_DeepCaT\_Z.py* file, and change it if necessary.

Make sure that the main parameters match the data properties that will be used for training and that the directories and model’s names are correct. The *segmentationModel\_functions.py* file contains models’ architectures and losses.

To run the training file (*main\_segmentationTask\_UNet.py*):

**For Windows:**

Click on the *main\_segmentation\_DeepCaT\_z.bat* file (inside the “TRAIN\_MODELS\SEGMENTATION” folder) and wait until the training is completed.



### **For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following line:

```
>> [insert the path of the Deep-CaT-z-Software-main folder]\TRAIN_MODELS\SEGMENTATION\  
main_segmentationTask_UNet.py
```

The best model with the best parameters will be saved after the training is over, and this model can then be used for automatic segmentation for controlling closed-loop behavioral experiments with the *DeepCaT-z* tool.

- Training the **classification** model

The *main\_classificationTask.py* file, inside the CLASSIFICATION folder, contains all functions needed for training the deep learning network for the automatic recognition of animal behavior. To edit the default parameters for the classification training operations, open any source code or *Python* editor (such as *Spyder* or *Notepad*), load the *config\_Classification\_DeepCaT\_Z.py* file, and change it if necessary.

Make sure that the main parameters match the data properties that will be used for training and that the directories and model's names are correct. The *classificationModel\_function.py* file contains models' architectures, utility functions, and losses.

To run the training file (*main\_classificationTask.py*),

### **For Windows:**

Click on the *main\_classification\_DeepCaT\_z.bat* file (inside the "TRAIN\_MODELS\CLASSIFICATION" folder) and wait until the training is completed.

### **For MAC:**

Open again the *Command Prompt* (see *Installation* steps for more details) and type the following line:

```
>> [insert the path of the Deep-CaT-z-Software-main folder]\TRAIN_MODELS\CLASSIFICATION\  
main_classificationTask.py
```

The model with the best parameters will be saved after the training is over, and this model can then be used for automatic behavior classification for controlling closed-loop behavioral experiments with the *DeepCaT-z* tool.

## *Contact Us*

For questions/comments, please send us an email to:  
[anafgeros@ineb.up.pt](mailto:anafgeros@ineb.up.pt) or [pauloaguiar@ineb.up.pt](mailto:pauloaguiar@ineb.up.pt).