Title

A Step-by-Step Implementation of DeepBehavior, Deep Learning Toolbox for Automated Behavior Analysis.

Authors

Shukla, Sanjay
Arac, Ahmet

Publication Date

2020

DOI

10.3791/60763

Peer reviewed

HHS Public Access

Author manuscript

J Vis Exp. Author manuscript; available in PMC 2021 February 06.

A Step-by-Step Implementation of DeepBehavior, Deep Learning Toolbox for Automated Behavior Analysis

Sanjay Shukla[1], **Ahmet Arac**[1]

[1]Department of Neurology, David Geffen School of Medicine, University of California, Los Angeles

Abstract

Understanding behavior is the first step to truly understanding neural mechanisms in the brain that drive it. Traditional behavioral analysis methods often do not capture the richness inherent to the natural behavior. Here, we provide detailed step-by-step instructions with visualizations of our recent methodology, DeepBehavior. The DeepBehavior toolbox uses deep learning frameworks built with convolutional neural networks to rapidly process and analyze behavioral videos. This protocol demonstrates three different frameworks for single object detection, multiple object detection, and three-dimensional (3D) human joint pose tracking. These frameworks return cartesian coordinates of the object of interest for each frame of the behavior video. Data collected from the DeepBehavior toolbox contain much more detail than traditional behavior analysis methods and provides detailed insights to the behavior dynamics. DeepBehavior quantifies behavior tasks in a robust, automated, and precise way. Following the identification of behavior, post-processing code is provided to extract information and visualizations from the behavioral videos.

Keywords

Behavior; Issue 156; Deep Learning; Behavior Analysis; Convolutional Neural Nets; Machine Learning; Kinematic Analysis; Automated Analysis; Animal Behavior; Human Behavior; Reaching Tasks; Image Data; Video Data; 3D Kinematics

Introduction

A detailed analysis of behavior is key to understanding the brain and behavior relationships. There have been many exciting advances in methodologies for recording and manipulating neuronal populations with high temporal resolution, however, behavior analysis methods have not developed at the same rate and are limited to indirect measurements and a reductionist approach[1]. Recently, deep learning based methods have been developed to perform automated and detailed behavior analysis[2,3,4,5]. This protocol provides a step-by-step implementation guide for the DeepBehavior toolbox.

Correspondence to: Ahmet Arac at aarac@mednet.ucla.edu.

Video Link

The video component of this article can be found at https://www.jove.com/video/60763/

Disclosures

The authors have nothing to disclose.

Traditional behavioral analysis methods often include manually labeling data by multiple evaluators, leading to variance in how experimenters define a behavior[6]. Manual labeling of the data requires time and resources that increase disproportionately to the amount of data collected. Moreover, manually labelled data reduce the behavior outcomes into categorical measurements which do not capture the richness of the behavior, and will be more subjective. Thus, the current traditional methods may be limited in capturing the details in the natural behaviors.

The DeepBehavior toolbox presents a precise, detailed, highly temporal, and automated solution using deep learning for behavioral analysis. Deep learning has quickly become accessible to all with open-source tools and packages. Convolutional neural networks (CNNs) are proven to be highly effective in object recognition and tracking tasks[7,8]. Using modern day CNNs and high-performance graphics-processing-units (GPUs), large image and video datasets can be processed quickly with high precision[7,9,10,11]. In DeepBehavior, there are three different convolutional neural net architectures, TensorBox, YOLOv3, and OpenPose[2].

The first framework, Tensorbox, is a versatile framework that incorporates many different CNN architectures for object detection[12]. TensorBox is best suited for detecting only one object class per image. The resulting outputs are bounding boxes of the object of interest (Figure 1) and the cartesian coordinates of the bounding box.

The second CNN framework is YOLOv3, which stands for “You Only Look Once”[13]. YOLOv3 is advantageous when there are multiple objects of interest that must be tracked separately. The output of this network includes the bounding box with the associated object label class as well as the bounding box cartesian coordinates of the object in the video frame (Figure 2).

The previous two frameworks are advantageous for generalized behavioral data collected from standard laboratory experiments in animal subjects. The last CNN framework is OpenPose[14,15,16] which is used for human joint pose estimation. OpenPose detects human body, hand, facial, and foot key points on images. The outputs of the framework are labeled images of the human subject as well as the coordinates of all the 25 key points in the body and 21 key points of each hand (Figure 3).

This detailed step-by-step guide for implementation of our recently developed open-source DeepBehavior toolbox employs state-of-the-art convolutional neural nets to track animal behavior (e.g. movement of a paw) or human behavior (e.g. reaching tasks). By tracking the behavior, useful kinematics can be derived from the behavior such as position, velocity, and acceleration. The protocol explains the installation of each CNN architecture, demonstrates how to create training datasets, how to train the networks, how to process new videos on the trained network, how to extract the data from the network on the new videos, and how to post-process the output data to make it useful for further analysis.

Protocol

1. GPU and Python Setup

1. GPU Software

   When the computer is first setup for deep learning applications, GPU-appropriate software and drivers should be installed which can be found on the GPU's respective website. (see the Table of Materials for those used in this study).

2. Python 2.7 Installation

   Open a command line prompt on your machine.

   Command line: sudo apt-get install python-pip python-dev python-virtualenv

2. TENSORBOX

1. Tensorbox Setup

   1. Create Virtual Environment for Tensorbox

      Command line: cd ~

      Command line: virtualenv --system-site-packages ~/tensorflow

      NOTE: '~/tensorflow' is the name of the environment and is arbitrary

   2. Activate environment

      Command line: source ~/tensorflow/bin/activate

2. Tensorbox Installation

   We will be using GitHub to clone TensorBox from http://github.com/aarac/TensorBox and install it on our machine as well as installing additional dependencies.

   Command line: cd ~

   Command line: git clone http://github.com/aarac/TensorBox

   Command line: cd TensorBox

   Command line: pip install -r requirements.txt

3. Label Data

   1. Create a folder of images of behavior

      Open source tools such as ffmpeg are useful to accomplish converting videos to individual frames We recommend labeling at least 600 images from a wide-distribution of behavior frames for training. Put these images in a folder.

   2. Launch labeling graphical user interface

      Command line: python make_json.py <path to image folder> labels.json

To label an image, click the top left corner of the object of interest (i.e. paw) first and then click the bottom right corner of the object of interest (Figure 4). Inspect that the bounding box captures the entire object of interest. Press 'undo' to re-label the same image or press 'next' to move onto the next frame.

4. Train TensorBox

    1. Link training images to network hyperparameters file

       Within the tensorbox folder, open the following folder in a text editor:

       /TensorBox/hypes/overfeat_rezoom.json. Navigate to the attribute under *data* named *train_idl* and replace the file path from *./data/brainwash/train_boxes.json* to the *labels.json* filepath. Save the changes to file.

    2. Begin training script

       Command line: cd ~/TensorBox

       Command line: python train.py --hypes hypes/overfeat_rezoom.json --gpu 0 --logdir output

       The network will then begin training for 600,000 iterations. In the output folder, the resulting trained weights of the convolutional neural network will be generated.

5. Predict on New Images

   For image labeling:

   Command line: cd ~/TensorBox

   Command line: python label_images.py --folder <path to image folder> --weights output/overfeat_rezoom_<timestamp>/save.ckpt-600000 --hypes /hypes/overfeat_rezoom.json --gpu 0

   To get coordinates of bounding boxes:

   Command line: cd ~/TensorBox

   Command line: python predict_images_to_json.py --folder <path to image folder> --weights

   output/overfeat_rezoom_<timestamp>/save.ckpt-600000 --hypes

   /hypes/overfeat_rezoom.json --gpu 0

6. MATLAB Post-Processing for TensorBox

   Additional MATLAB code has been provided to extract kinematics and visualizations of the coordinates using the resulting JSON coordinate file from the model

   Run the "Process_files_3Dreaching_mouse.m" script for 3D kinematic analysis of single food pellet reaching task.

3. YOLOv3

1. Install YOLOv3

   Command line: cd ~

   Command line: git clone cd *darknet*

   For GPU usage, open 'Makefile' and change the following lines: GPU=1; CUDNN=1.

   Command line: make

2. Labeling Training Data using Yolo_mark

   Command line: cd ~

   Command line: git clone cd ~/Yolo_Mark

   Command line: cmake .

   Command line: make

   Place the training images in *~/Yolo_mark/data/obj* folder

   Command line: chmod +x ./linux_mark.sh

   Command line: ./linux_mark.sh

   Label the images one by one in the graphical user interface (Figure 5). The recommended amount of images is approximately 200.

3. Training YOLOv3

   1. Setup configuration file

      Command line: cd ~/Yolo_mark

      Command line: scp -r ./data ~/darknet

      Command line: cd ~/darknet/cfg

      Command line: cp yolov3.cfg yolo-obj.cfg

   2. Modify the configuration file

      Open the yolo-obj.cfg folder and modify the following lines: batch=64, subdivision=8, classes=(# of class to detect), and for each convolutional layer before a yolo layer change the filter=(classes+5)x3. Details on these changes can be found at https://github.com/aarac/darknet/blob/master/README.md

   3. Download network weights

      Download the network weights from https://www.dropbox.com/s/613n2hwm5ztbtuf/darknet53.conv.74?dl=0

      Place the downloaded weight file into *~/darknet/build/darknet/x64*

   4. Run training algorithm

Command line: cd ~/darknet

Command line: ./darknet detector train data/obj.data cfg/yolo-obj.cfg darknet53.conv.74

5. YOLOv3 Evaluation

   After the training is complete based on a set number of iterations (*ITERATIONNUMBER)*, you can view them by

   Command line: ./darknet detector test data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_ITERATIONNUMBER.weights <IMAGE>.jpg

4. Predict on new videos and get coordinates

   This command can be run to obtain the coordinates of the labels in the new video:

   Command line: ./darknet detector demo data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_ITERATIONNUMBER.weights VIDEO.avi -ext_output <VIDEO.avi> FILENAME.txt

5. YOLOv3 PostProcessing in MATLAB

   Take the FILENAME.txt file to MATLAB, and run the "Process_socialtest_mini.m" script for two mice social interaction test. See results in Figure 2

4. OpenPose

OpenPose is ideal to track multiple body parts in a human subject. The setup and installation processes are very similar to the previous two frameworks. However, there is no training step as the network is already trained on human data.

1. OpenPose Installation

   Navigate to https://github.com/aarac/openpose and follow the installation instructions.

2. Process Video

   ./build/examples/openpose/openpose.bin --video VIDEONAME.avi --net_resolution "1312×736" --scale_number 4 --scale_gap 0.25 --hand --hand_scale_number 6 --hand_scale_range 0.4 --write_json JSONFOLDERNAME --write_video RESULTINGVIDEONAME.avi

   Here the --net_resolution, --scale_number, --scale_gap, --hand_scale_number and --hand_scale_range handles can be omitted if a high precision detection is not needed (this would decrease the processing time).

3. OpenPose Post-Processing

   In MATLAB folder, please use 'process_files_human3D.m' script to run the code after adding the appropriate folder containing json files from cameras 1 and 2, as well as the calibration file. This will create a "cell" file with all the 3D

poses of the joints. It will also make a movie of the 3D skeletal view. For camera calibration, please follow the instructions at this link: http://www.vision.caltech.edu/bouguetj/calib_doc/

Representative Results

When the protocol is followed, the data for each network architecture should be similar to the following. For TensorBox, it outputs a bounding box around the object of interest. In our example, we used videos from a food pellet reaching task, and labeled the right paws to track their movement. As seen in Figure 1, the right paw can be detected in different positions in both the front view and side view cameras. After post-processing with camera calibration, 3D trajectories of the reach can be obtained (Figure 1B).

In Yolov3, as there are multiple objects, the output is also multiple bounding boxes. As seen in Figure 2B, there are multiple bounding boxes around the objects of interest. These can be parts of the body.

In OpenPose, the network detects the joint positions as seen in Figure 3A. After post-processing with camera calibration, a 3D model of the subject can be created (Figure 3B).

In conclusion, these representative results showcase the rich details of behavior that can be captured using the DeepBehavior toolbox.

Discussion

Here, we provide a step-by-step guide for implementation of DeepBehavior, our recently developed deep learning based toolbox for animal and human behavior imaging data analysis[2]. We provide detailed explanations for each step for installation of the frameworks for each network architecture, and provide links for installation of the open-source requirements to be able to run these frameworks. We demonstrate how to install them, how to create training data, how to train the network, and how to process new video files on the trained network. We also provide the post-processing code to extract the basic necessary information needed for further analysis.

For single object detection, we recommend using TensorBox. If the goal is to track multiple objects at once, we recommend using YOLOv3. Finally, to obtain human kinematic data, we recommend using OpenPose. In this protocol we have shown that deep learning methods are able to process hundreds of thousands of frames while tracking objects with a high degree of precision. Using the post-processing code provided, we can derive meaningful ways of analyzing the tracked behavior of interest. This provides a more detailed way of capturing behavior. It also provides an automated, robust way of defining behavior that is generalizable to many different types of behavioral tasks.

It is quite common to get a ‘ModuleNotFoundError’ when starting with a new virtual environment or code that has been downloaded from the internet. In the case that this occurs, open up your terminal, activate the source environment and type *‘pip install <missing*

module name>'. If the problem persists, you will need to check your python version as well as other dependency packages.

Limitations to this technique include the technical troubleshooting to properly set up GPU processing units compatible with open-source code. It is advantageous to have past programming experience within a linux environment to properly set up the necessary project dependencies and environments that are compatible with the computer's hardware.

We demonstrate the DeepBehavior toolbox installations and processing of in a linux environment, however, this toolbox can also be run on a Windows and Mac machines with GPUs by following the respective installation guides on github.

Using deep learning methods for imaging data analysis is a very efficient way to automate behavior analysis. In comparison to traditional behavior analysis methods, DeepBehavior captures much more information to quantify, automate, and evaluate the behavior at a more precise and temporally detailed way. With the further advances in the deep learning field, the utilization and extent of the use of this technology in behavior analysis will likely continue to improve. The applications of DeepBehavior can be expanded beyond the demonstrated reaching tasks to identify objects of interest in any behavioral images. In this protocol, we provide detailed instructions to implement three neural networks for behavior analysis. With this kind of automated and unbiased behavior analysis methods, hopefully, the neuroscience field will be able to do more detail behavior analysis.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

We would like to thank Pingping Zhao and Peyman Golshani for providing the raw data for two-mouse social interaction tests used in the original paper[2]. This study was supported by NIH NS109315 and NVIDIA GPU grants (AA).

References

1. Krakauer JW, Ghazanfar AA, Gomez-Marin A, MacIver MA, Poeppel D Neuroscience Needs Behavior: Correcting a Reductionist Bias. Neuron. 93 (3), 480–90 (2017). [PubMed: 28182904]
2. Arac A, Zhao P, Dobkin BH, Carmichael ST, Golshani P DeepBehavior: A Deep Learning Toolbox for Automated Analysis of Animal and Human Behavior Imaging Data. Front Syst Neurosci. 13, 20 (2019). [PubMed: 31133826]
3. Pereira TD, Aldarondo DE, Willmore L, Kislin M, Wang SS, Murthy M, et al. Fast animal pose estimation using deep neural networks. Nat Methods. 16 (1), 117–25 (2019). [PubMed: 30573820]
4. Mathis A, Mamidanna P, Cury KM, Abe T, Murthy VN, Mathis MW, et al. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nat Neurosci. 21 (9), 1281–9 (2018). [PubMed: 30127430]
5. Stern U, He R, Yang CH Analyzing animal behavior via classifying each video frame using convolutional neural networks. Sci Rep. 5, 14351 (2015). [PubMed: 26394695]
6. Tinbergen N On aims and methods of ethology. Zeitschrift fü r Tierpsychologie. 20, 410–33 (1963).
7. LeCun Y, Bengio Y, Hinton G Deep learning. Nature. 521 (7553), 436–44 (2015). [PubMed: 26017442]

8. Zhao Z, Zheng P, Xu S, Wu X Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems. 1–21 (2019).
9. He K, Zhang X, Ren S, Sun J Deep Residual Learning for Image Recognition. arXiv. eprint (2015).
10. Krizhevsky A, Sutskever I, Hinton GE ImageNet classification with deep convolutional neural networks. Proceedings of the 25th International Conference on Neural Information Processing Systems Lake Tahoe, Nevada Curran Associates Inc. 1097–1105 (2012).
11. Szegedy C, Wei L, Yangqing J, Sermanet P, Reed S, Anguelov D, et al., editors. Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 7–12 6 (2015).
12. Stewart R, Andriluka M, Ng AY, editors. End-to-End People Detection in Crowded Scenes. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 27–30 6 (2016).
13. Redmon J, Farhadi A YOLOv3: An Incremental Improvement. arXiv. eprint (2018).
14. Cao Z, Simon T, Wei S-E, Sheikh Y Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. arXiv. (2017).
15. Simon T, Joo H, Matthews I, Sheikh Y Hand Keypoint Detection in Single Images using Multiview Bootstrapping. arXiv. eprint (2017).
16. Wei S-E, Ramakrishna V, Kanade T, Sheikh Y Convolutional Pose Machines. arXiv. eprint (2016).

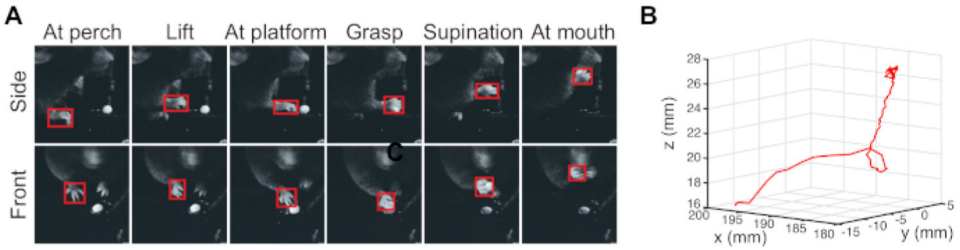

Figure 1: Bounding boxes with TensorBox seen on the paws of video frames during a reaching task in mice.

(Adapted from Arac et al 2019). Please click here to view a larger version of this figure.

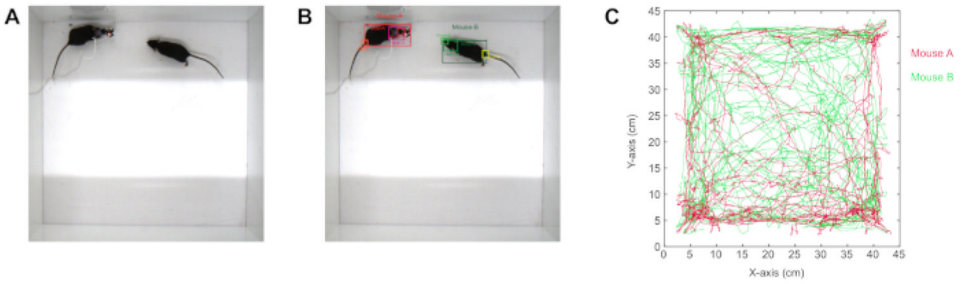

Figure 2: Bounding boxes with Yolov3 seen on the regions of interest in video frames during a two mice social interaction test (A raw image, B analyzed image).
(Adapted from Arac et al 2019). Please click here to view a larger version of this figure.

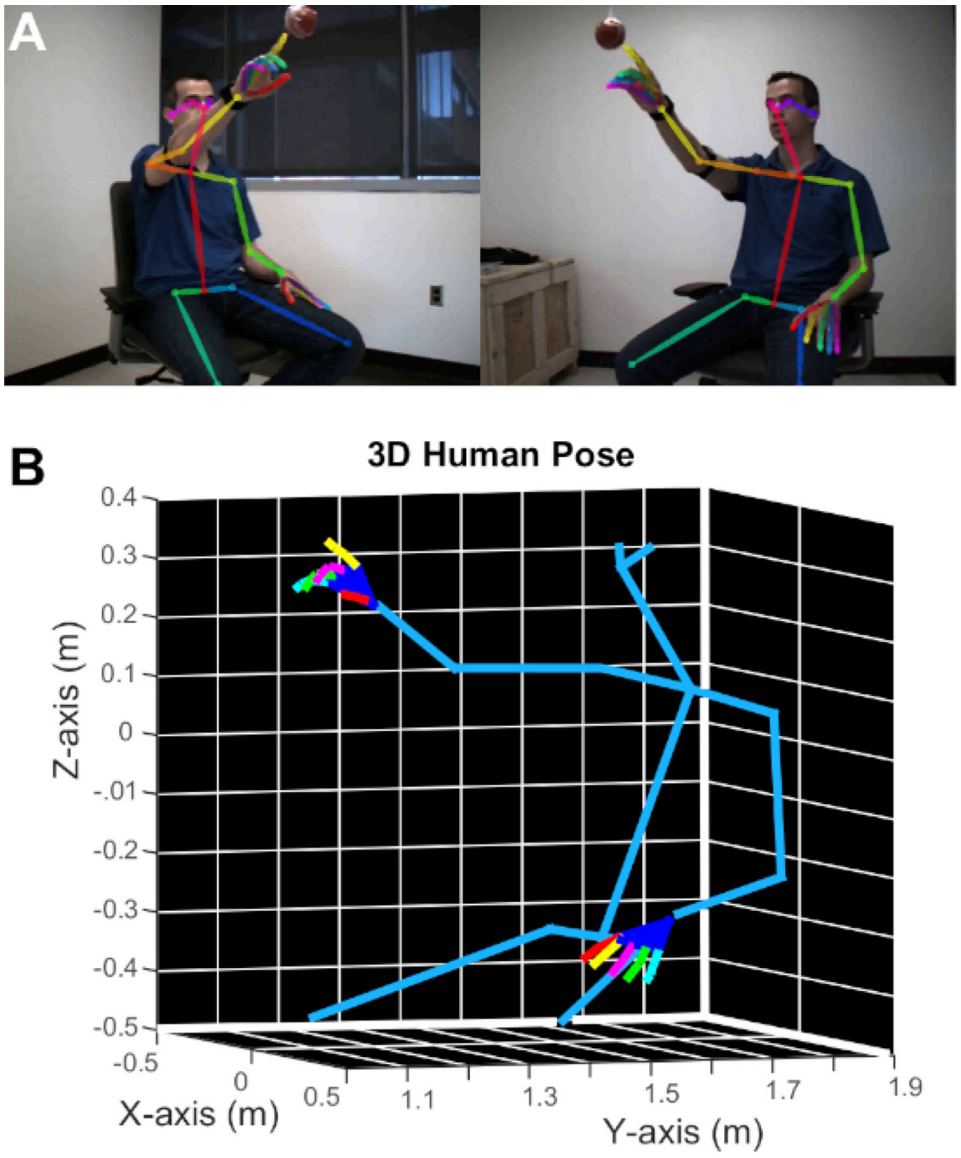

Figure 3: Human pose detection with OpenPose in two camera views (A) and 3D model created from these two images (B).

(Adapted from Arac et al 2019). Please click here to view a larger version of this figure.

Figure 4: TensorBox's make_json GUI used to label training data.
Please click here to view a larger version of this figure.

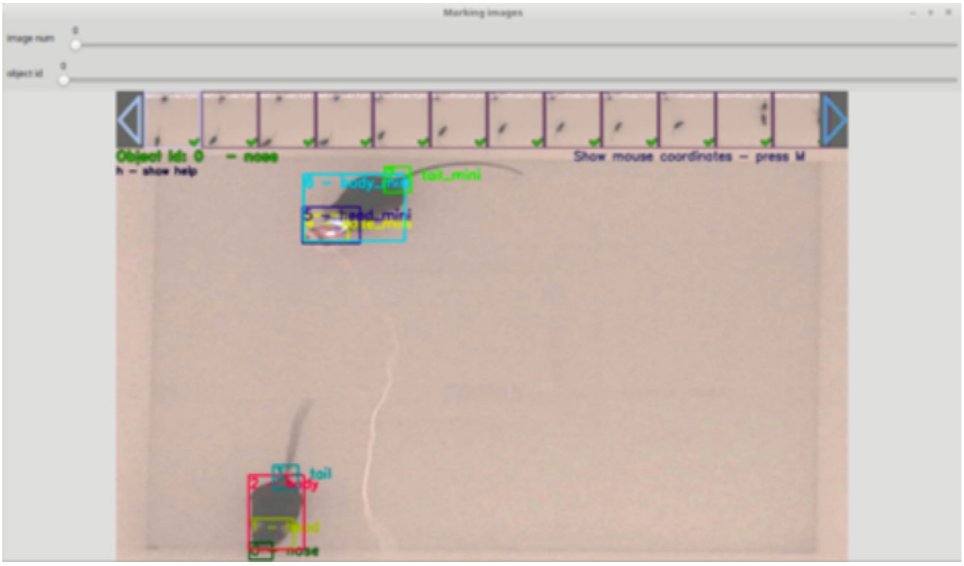

Figure 5: GUI of Yolo_Mark to label images in a format acceptable for Yolov3.
Please click here to view a larger version of this figure.