nature protocols

PROTOCOL

https://doi.org/10.1038/s41596-019-0176-0

Using DeepLabCut for 3D markerless pose estimation across species and behaviors

Tanmay Nath[1,5], Alexander Mathis[1,2,5], An Chi Chen[3], Amir Patel[3], Matthias Bethge[4] and Mackenzie Weygandt Mathis[1*]

Noninvasive behavioral tracking of animals during experiments is critical to many scientific pursuits. Extracting the poses of animals without using markers is often essential to measuring behavioral effects in biomechanics, genetics, ethology, and neuroscience. However, extracting detailed poses without markers in dynamically changing backgrounds has been challenging. We recently introduced an open-source toolbox called DeepLabCut that builds on a state-of-the-art human pose-estimation algorithm to allow a user to train a deep neural network with limited training data to precisely track user-defined features that match human labeling accuracy. Here, we provide an updated toolbox, developed as a Python package, that includes new features such as graphical user interfaces (GUIs), performance improvements, and active-learning-based network refinement. We provide a step-by-step procedure for using DeepLabCut that guides the user in creating a tailored, reusable analysis pipeline with a graphical processing unit (GPU) in 1–12 h (depending on frame size). Additionally, we provide Docker environments and Jupyter Notebooks that can be run on cloud resources such as Google Colaboratory.

Introduction

Advances in computer vision and deep learning are transforming research. Applications such as those for autonomous car driving, estimating the poses of humans, controlling robotic arms, or predicting the sequence specificity of DNA- and RNA-binding proteins are being rapidly developed[1–4]. However, many of the underlying algorithms are 'data hungry', as they require thousands of labeled examples, which potentially prohibits these approaches from being useful for small-scale operations, such as in single-laboratory experiments[5]. Yet transfer learning, the ability to take a network that was trained on a task with a large supervised dataset and utilize it for another task with a small supervised dataset, is beginning to allow users to broadly apply deep-learning methods[6–10]. We recently demonstrated that, owing to transfer learning, a state-of-the-art human pose–estimation algorithm called DeeperCut[10,11] could be tailored for use in the laboratory with relatively small amounts of annotated data[12]. Our toolbox, DeepLabCut[12], provides tools for creating annotated training sets, training robust feature detectors, and utilizing them to analyze novel behavioral videos. Current applications encompass common model systems such as mice, zebrafish, and flies, as well as rarer ones such as babies and cheetahs (Fig. 1).

Here, we provide a comprehensive protocol and expansion of DeepLabCut that allows researchers to estimate the pose of a subject, efficiently enabling them to quantify behavior. We have previously provided a version of this protocol on BioRxiv[13]. The major motivation for developing the DeepLabCut toolbox was to provide a reliable and efficient tool for high-throughput video analysis, in which powerful feature detectors of user-defined body parts need to be learned for a specific situation. The toolbox is aimed to solve the problem of detecting body parts in dynamic visual environments in which varying background, reflective walls, or motion blur hinder the performance of common techniques such as thresholding or regression based on visual features[14–21]. The uses of the toolbox are broad; however, certain paradigms will benefit most from this approach. Specifically, DeepLabCut is best suited to behaviors that can be consistently captured by one or multiple cameras with minimal occlusions. DeepLabCut performs frame-by-frame prediction and thereby can also be used for analysis of behaviors with intermittent occlusions.

[1]Rowland Institute at Harvard, Harvard University, Cambridge, MA, USA. [2]Department of Molecular & Cellular Biology, Harvard University, Cambridge, MA, USA. [3]Department of Electrical Engineering, University of Cape Town, Cape Town, South Africa. [4]Tübingen AI Center & Centre for Integrative Neuroscience, Eberhard Karls Universität Tübingen, Tübingen, Germany. [5]These authors contributed equally: Tanmay Nath, Alexander Mathis. *e-mail: mackenzie@post.harvard.edu

Fig. 1 | Pose estimation with DeepLabCut. Six examples of DeepLabCut-applied labels. The colored points represent features the user wished to measure. **a**–**f**, Examples of a fruit fly (**a**), a cheetah (**b**), a mouse hand (**c**), a horse (**d**), a zebrafish (**e**), and a baby (**f**). **c** reproduced with permission from Mathis et al.[12], Springer Nature. **f** reproduced with permission from Wei and Kording[26], Springer Nature.

The DeepLabCut Python toolbox is versatile, easy to use, and does not require extensive programming skills. With only a small set of training images, a user can train a network to within human-level labeling accuracy, thus expanding its application to not only behavior analysis in the laboratory, but potentially also to sports, gait analysis, medicine, and rehabilitation studies[14,22–26].

Overview of using DeepLabCut

DeepLabCut is organized according to the following workflow (Fig. 2). The user starts by creating a new project based on a project name and username as well as some (initial) videos, which are required to create the training dataset (Stages I and II). Additional videos can also be added after the creation of the project, which will be explained in greater detail below. Next, DeepLabCut extracts frames that reflect the diversity of the behavior with respect to, e.g., postures and animal identities (Stage III). Then the user can label the points of interest in the extracted frames (Stage IV). These annotated frames can be visually checked for accuracy and corrected, if necessary (Stage V). Eventually, a training dataset is created by merging all the extracted labeled frames and splitting them into subsets of test and train frames (Stage VI). Then a pre-trained network (ResNet) is refined end-to-end to adapt its weights in order to predict the desired features (i.e., labels supplied by the user; Stage VII). The performance of the trained network can then be evaluated on the training and test frames (Stage VIII). The trained network can be used to analyze videos, yielding extracted pose files (Stage IX). If the trained network does not generalize well to unseen data in the evaluation and analysis step (Stages VIII and IX), then additional frames with poor results can be extracted (optional Stage X), and the predicted labels can be manually corrected. This refinement step, if needed, creates an additional set of annotated images that can then be merged with the original training dataset. This larger training set can then be used to re-train the feature detectors for better results. This active-learning loop can be done iteratively to robustly and accurately analyze videos with potentially large variability, i.e., experiments that include many individuals and run over long time periods. Furthermore, the user can add additional body parts/labels at later stages during a project, as well as correct user-defined labels. We also provide Jupyter Notebooks, a summary of the commands, a section on how to work with the output files (Stage XI), and a 'Troubleshooting' section.

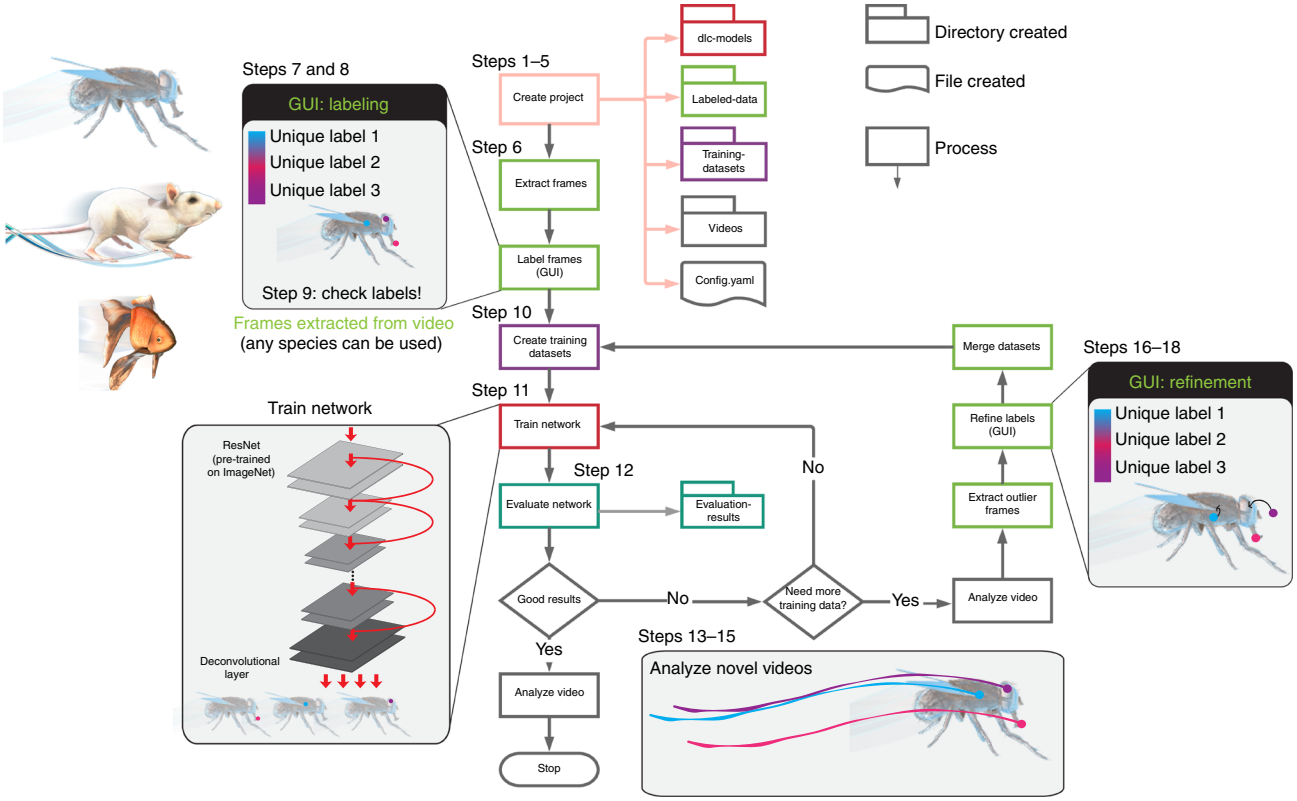

Fig. 2 | DeepLabCut workflow. The diagram delineates the workflow, as well as the directory and file structures. The process of the workflow is color coded to represent the locations of the output of each step or section. The main steps are opening a Python session, importing DeepLabCut, creating a project, selecting frames, labeling frames, and training a network. Once trained, this network can be used to apply labels to new videos, or the network can be refined if needed. The process is fueled by interactive graphical user interfaces (GUIs) at several key steps.

Applications

DeepLabCut has been used to extract user-defined, key body part locations from animals (including humans) in various experimental settings. Current applications range from tracking mice in open-field behaviors to more complicated hand articulations in mice and whole-body movements in flies in a 3D environment (all shown in Mathis et al.[12]) as well as pose estimation in human babies[26], 3D locomotion in rodents[27], and multi-body-part tracking (including eye tracking) during perceptual decision making[28], in horses and in cheetahs (Fig. 1). Other (currently unpublished) use cases can be found at http://www.deeplabcut.org. Finally, 3D pose estimation for a challenging cheetah hunting example is presented in this article.

Comparison with other methods

Pose estimation is a challenging, yet classic, computer vision problem[29] whose human pose–estimation benchmarks have recently been shattered by deep-learning algorithms[2,10,11,30–33]. There are several main considerations when deciding which deep-learning algorithms to use: namely, the amount of labeled input data required, the speed, and the accuracy. DeepLabCut was developed to require minimal labeled data, to allow for real-time processing (i.e., as fast or faster than camera acquisition), and to be as accurate as human annotators.

DeepLabCut utilizes the feature detectors of an algorithm called DeeperCut, which is one of the best algorithms for several human pose–estimation benchmarks[10]. Specifically, DeepLabCut currently uses deep, residual networks with either 50 or 101 layers (ResNets)[34] and deconvolutional layers as developed in DeeperCut[10]. As pose estimation in the lab is typically simpler than most benchmarks in computer vision, we were able to remove aspects of DeeperCut that were not required to achieve human-level accuracy on several laboratory tasks[12]. The removal of the pairwise refinement, as well as integer linear programming on top of the part detectors, substantially increased the inference speed

(full multi-human DeeperCut: 578–1,171 s per frame vs. DeepLabCut: 10–90 frames per second (FPS)). Furthermore, we recently implemented faster inference[27] (see below).

If a user aims to track (adult) human poses, many other excellent options exist, including DeeperCut, ArtTrack, DeepPose, OpenPose, and OpenPose-Plus[2,10,11,30–33,35]. Some of these methods also allow real-time inference. To our knowledge, the performance of these methods has not been investigated on non-human animals. Conversely, because the trained networks can be used, they provide excellent tools for pose estimation of humans. There are also specific networks for faces[32] and hands[33]. However, if (additional) body parts beyond the body parts contained in the pre-trained networks need to be labeled, then DeepLabCut could also be useful (and we provide download links for a network pre-trained on humans). Recently, another neural network–based package for animals, called LEAP, was described[36]. LEAP excels at rapidly training a network on a specific behavior, but because it is a shallow network without any pre-training, it is probably not as robust to changes in the environment and requires more training data to match the performance of DeepLabCut, i.e., human-level accuracy. Although the authors highlight an interactive training framework (which can be done with DeepLabCut, if desired) and suggest starting with tens of frames, it needs ~500 images to achieve <3 pixel error on 192 × 192-pixel frames[36]. DeepLabCut achieves <5 pixel error with 100 labeled frames on 800 × 800-pixel-sized dataset and reaches human-level accuracy of 2.7 pixels with 500 labeled frames. These advantages seem to be due to transfer learning[12], whereas LEAP is trained from scratch. DeepLabCut was shown to need <150 frames of human-labeled data for even challenging 3D hand movements of a mouse[12]. DeepLabCut was also shown to generalize to novel, and multiple, individuals, as well as to multiple conditions in dynamic environments[12]. Inference speed (i.e., the runtime of the trained network on novel videos) may also be a consideration for users. DeepLabCut can process 10–90 FPS (depending on the frame size, and if the batch size is 1, as is required for low-latency uses), meaning that it achieves real-time processing if the camera speed is 10–90 FPS. By contrast, LEAP may be faster, because of its 15-layer convolutional network, as the authors report a processing speed of 185 FPS with a batch size of 32 images for 192 × 192-sized images[36]. For similarly sized videos, DeepLabCut runs at 90 FPS with a batch size of 1 and ~350 FPS with a batch size of 32, and accelerates substantially (up to 1,000 FPS) for larger batch sizes[27].

Advantages

DeepLabCut has been applied to a range of organisms with diverse visual background challenges. The main advantages are that our code (i) guides the experimenter with a step-by-step procedure from labeling data to automated pose extraction in a fast and efficient way (Figs. 2–6), (ii) minimizes the cost of manual behavior analysis and, with only a small number of training images, achieves human-level accuracy, (iii) eliminates the need to put visible markers on the locations of interest, (iv) can be easily adapted to analyze behaviors across species, and (v) is open source and free. Owing to the use of deep features, DeepLabCut can learn to robustly extract body parts, even with a cluttered and varying background, inhomogeneous illumination, or camera distortions[12]. Thus, experimenters can perhaps design studies more around their scientific question, rather than the constraints imposed by previous tracking algorithms. For example, it is common to image mice on a plain white, gray, or black background to provide contrast with their coat color. Now, even natural backgrounds, such as home-cage bedding and natural grass, or dynamically changing backgrounds, such as in trail tracking[12], can be used. This is well illustrated by the cheetah example provided in this paper (Figs. 1 and 7; Supplementary Video 1).

DeepLabCut also allows for 3D pose estimation via multi-camera use. A user can train a network for each camera view, or combine multiple camera views and train one network that generalizes across all views (see Fig. 7 below and ref. [27]) and then use standard camera calibration techniques[37] to resolve 3D locations. DeepLabCut does not require images to have a fixed frame size, as the feature detectors are not sensitive (within bounds) to the size because of automatic rescaling during training[12]. There are also no specific camera requirements. Color and grayscale images captured from scientific cameras or consumer-grade cameras such as a GoPro can be used (and the package supports multiple video types, such as .mp4 and .avi). Also, we recently showed that DeepLabCut is robust to video compression, potentially saving users >500-fold on data storage space[27], making multi-camera use potentially more feasible.

The toolbox provides different frame-extraction methods to accommodate the analysis of videos from varying recording sessions. The toolbox is modular, and by changing desired parameters at

execution, the user can, for instance, utilize different frame-extraction methods or identify frames that require further inspection. Reliably defining the user-defined labels across different frames is of paramount importance for supervised learning approaches such as DeepLabCut. The labels can be body parts or any other readily visible object or part thereof. The toolbox also provides interactive tools (GUIs) to create, edit, and even add additional, new labels at a later stage of the project. The labels are stored in human-readable and efficient data structures. The code generates output files/directories for essential intermediate and final results in an automatic fashion. We believe that the error messages are intuitive and that they enable researchers to efficiently utilize the toolbox. Furthermore, the toolbox facilitates visualization of the labeled data and creates videos with the extracted labels overlaid, making the entire toolbox a complete package for behavioral tracking. The extracted poses per video can be further analyzed with Python or imported into many other programs for further analysis.

Limitations

One main limitation is that the toolbox requires modern computational hardware to produce fast and efficient results (namely, GPUs[5]). However, it is possible to run this toolbox on a standard computer (only CPUs) with a compromise on the speed of analysis; it is ~10–100× slower[27]. However, the availability of inexpensive consumer-grade GPUs has opened the door for labs to perform advanced image processing in an autonomous way (i.e., in the lab, without lengthy transfer of data to centralized computer clusters).

Another consideration is that deep convolutional networks scale with the size of pixels; therefore, larger images will be processed more slowly. Thus, for applications that would benefit from rapid analysis of the input, images should be downsampled to achieve high rates (i.e., 90 FPS for a frame size of 138 × 138 with a batch size of 1). However, with new code updates and batch-processing options, one can rapidly increase the analysis speed. We recently demonstrated that with a batch size of 64, for a frame size of ~138 × 138, one can achieve 600 FPS[27].

Another limitation is that DeepLabCut, because it is designed to be of general purpose, does not rely on heuristics such as a body model, and therefore occluded points cannot be tracked. However, DeepLabCut outputs a confidence score, which reports if a body part is actually visible. As we showed, one can (for instance) tell which legs are visible in a fly moving through a 3D space, or whether the fingertips of a reaching mouse are grasping the joystick and are thus occluded[12]. Furthermore, because DeepLabCut performs frame-by-frame prediction, even if—due to occlusions, motion blur, or another reason— features cannot be detected in a few frames, they will be detected as soon as they are visible (unlike in many (actual) tracking methods, which require consistency across frames, such as the widely used Lucas–Kanade method[38]).

Materials

Equipment

Software

- Operating system: Linux (Ubuntu 16.04 LTS, 18.04 LTS), Windows (10), or MacOS
- Anaconda, a free and open-source distribution of the Python programming language (https://www.anaconda.com/). DeepLabCut is written in Python 3.6.x (https://www.python.org/) and is not compatible with Python 2
- DeepLabCut: the actively maintained toolbox is freely available at https://github.com/AlexEMG/DeepLabCut. The code is written for Python 3.6 (ref. [39]) and TensorFlow[40] for the feature detectors[10]
- TensorFlow[40], an open-source software library for Deep Learning. The toolbox is tested with TensorFlow v.1.0–1.4, 1.8, and 1.10–1.13. Any of these versions can be installed from https://www.tensorflow.org/install/
- (Optional) Docker[41]; we recommend using the supplied Docker container, which includes DeepLabCut and TensorFlow with GPU support pre-installed. This container builds on the nvidia-docker, which is currently supported only in Ubuntu
- (Optional) Jupyter Notebooks: we provide three Jupyter Notebooks for using DeepLabCut with two pre-labeled datasets and one template notebook for user datasets. First, we prepared an interactive Jupyter Notebook called *run_yourowndata.ipynb* that can serve as a template when developing a project. We provide two notebooks for an already-started project with labeled data. The example project, named Reaching-Mackenzie-2018-08-30, consists of a project configuration file with default

parameters and 19 images, which are cropped around the region of interest as an example dataset. These images are extracted from a video that was recorded in a study of skilled motor control in mice[42]. Furthermore, we provide another example project (openfield-Pranav-2018-10-30) with 116 labeled images from a trail-tracking mouse[12]. Details on using these Notebooks can be found at https://github.com/AlexEMG/DeepLabCut/tree/master/examples

- Dataset. With the above-mentioned Jupyter Notebooks we provide pre-labeled data for a mouse in an open-field-like arena (this report, and see ref. [12]) and a pre-labeled reaching dataset (adapted from ref. [42]). To use the supplied data, you must download the code from GitHub: https://github.com/AlexEMG/DeepLabCut. The cheetah data are available upon reasonable request

Hardware

- Computer; any modern desktop workstation will be sufficient, as long as it has a PCI slot as well as sufficient power supply for a GPU (see next item). The toolbox can also be used on laptops (e.g., for labeling data); then training can occur either on a CPU or elsewhere with a GPU. Note that training/evaluation of the feature detectors will be slow without a GPU, but it is possible[27]. We recommend 32 GB of RAM on the system for CPU analysis, but this is not a hard minimum. More information on optimally running TensorFlow can be found at https://www.tensorflow.org/guide/performance/overview
- GPU; we recommend using a GPU with at least 8 GB of memory, such as the NVIDIA GeForce 1080 or 2080. However, a CPU is sufficient, but training/evaluation of the network steps is considerably slower[27]. GPUs with less memory might also work well. This toolbox can also be used on cloud computing services (such as Google Cloud/Amazon Web Services)
- Camera(s); the toolbox is robust to extraction of poses from videos collected by many cameras. There are no a priori limitations in terms of lighting; color or grayscale images are acceptable, as are videos captured under infrared light, and inhomogeneous or natural lighting can be used. Cameras should be placed such that the features the user wishes to track are visible (to the user). For reference, as reported in Mathis et al.[12], we have used the following cameras for video capture: Firefly (Point Grey, model no. FMVU-03MTM-CS), Grasshopper3 4.1 MP Mono USB3 Vision (Point Grey, model no. CMOSIS CMV4000-3E12), or an infrared-sensitive CMOS camera from Basler. Here, the cheetahs were filmed with Hero5 Session cameras (GoPro, model no. Hero5), and industrial cameras (The Imaging Source, model no. DFK-37BUX287) were used to film mice

Equipment setup

Installation

It takes ~10–60 min to install the toolbox, depending on the installation method chosen.

We recommend that users first simply install DeepLabCut in an Anaconda environment (we provide the environment files at www.deeplabcut.org). For GPU support, users can either set up TensorFlow, CUDA, and the NVIDIA card driver on their own computers (following NVIDIA and TensorFlow's operation system– and graphics card–specific instructions; see below for links) or, alternatively, they can use our supplied Docker container, which has TensorFlow and CUDA pre-installed, or Google Colaboratory Notebooks that provide GPU access on the cloud.

A caveat to keep in mind is that by running '`pip install deeplabcut`' outside of an environment, all required distributions are installed (except wxPython and TensorFlow). If you perform such a system-wide installation, and the computer has other Python libraries, this will overwrite them. If you have a dedicated machine for DeepLabCut, this is fine. If there are other applications that require different versions of libraries, then you would potentially break those applications. One solution is to create a 'virtual environment', a self-contained directory that contains a Python installation for a particular version of Python, plus additional packages. One way to manage virtual environments is to use conda environments (for which you need Anaconda installed).

All the following commands will be run in a command-line interpreter ('terminal' in Ubuntu and MacOS and 'cmd' in Windows). Please first open the terminal (search 'terminal' or 'cmd').

If you wish to create your own environment (especially for Ubuntu users), it can be created by typing the following command into the terminal:

```
conda create -n <name_of_the_environment> python=3.6
```

This environment can then be accessed by typing the following:

In Windows: `activate <name_of_the_environment>`

In Linux: `source activate <name_of_the_environment>`

Once the environment is activated, the user can install DeepLabCut as described below (note that these steps are not required inside the environment files we provide at www.deeplabcut.org). A user can exit the conda environment at any time by typing the following command into the terminal:

In Windows: `deactivate <name_of_the_environment>`

In Linux: `source deactivate <name_of_the_environment>`

The user can reactivate the environment as shown above. The toolbox is installed within the environment and once users deactivates it, they must re-enter the environment using the DeepLabCut toolbox.

To install DeepLabCut, type the following command into the terminal:

```
pip install deeplabcut
```

For GUI support, type the following into the terminal:
Windows: `pip install -U wxPython`

Linux (Ubuntu 16.04): `pip install https://extras.wxpython.org/wxPython4/extras/linux/gtk3/ubuntu-16.04/wxPython-4.0.3-cp36-cp36m-linux_x86_64.whl`

As users may vary in their use of a GPU or a CPU, TensorFlow is not installed with the command `pip install deeplabcut`. For CPU-only support, type the following into the terminal:

```
pip install tensorflow==1.10
```

If you have a GPU, you should then install the NVIDIA CUDA package and an appropriate driver for your specific GPU. Follow the instructions at https://www.tensorflow.org/install/gpu, as we cannot cover all the possible combinations that exist (and these are continually updated packages).

DeepLabCut Docker

When using a GPU, we recommend using the supplied Docker container if you use Ubuntu, as TensorFlow and DeepLabCut are already installed. Unfortunately, a dependency (nvidia-docker) is currently not compatible with Windows. In addition, the GUIs typically are not used inside Docker containers. We envision a user installing DeepLabCut into an Anaconda environment but then executing the GPU steps inside the container. This way, installation is extremely simple. Full details on installing Docker can be found at https://github.com/MMathisLab/Docker4DeepLabCut2.0.

Last, it is also possible to run the network training steps of DeepLabCut in the supplied Docker, or on cloud computing resources (such as AWS, Google, or a University cluster). For running DeepLabCut on certain platforms you will need to suppress the GUI support. This can be done by setting an environment variable before loading the toolbox:

Linux: `export DLClight=True`

Windows: `set DLClight=True`

If you want to re-engage the GUIs (and have installed wxPython as described above):

Linux: `unset DLClight`

Windows: `set DLClight=`

Table 1 | Summary of commands

Operation	Command
Open IPython and import DeepLabCut (Step 1)	`ipython` `import deeplabcut`
Create a new project (Step 2)	`deeplabcut.create_new_project('project_name','experimenter',['path of video 1','path of video2',...])`
Set a `config_path` variable for ease of use (Step 3)	`config_path = '/yourdirectory/project_name/config.yaml'`
Extract frames (Step 4)	`deeplabcut.extract_frames(config_path)`
Label frames (Steps 5 and 6)	`deeplabcut.label_frames(config_path)`
Check labels (optional)(Step 7)	`deeplabcut.check_labels(config_path)`
Create training dataset (Step 8)	`deeplabcut.create_training_dataset(config_path)`
Train the network (Step 9)	`deeplabcut.train_network(config_path)`
Evaluate the trained network (Step 11)	`deeplabcut.evaluate_network(config_path)`
Video analysis and plotting results (Step 11)	`deeplabcut.analyze_videos(config_path, ['path of video 1 or folder','path of video2',...])`
Video analysis and plotting results (Step 12)	`deeplabcut.plot_trajectories(config_path, ['path of video 1','path of video2',...])`
Video analysis and plotting results (Step 13)	`deeplabcut.create_labeled_video(config_path, ['path of video 1','path of video2',...])`
Refinement: extract outlier frames (Step 14)	`deeplabcut.extract_outlier_frames(config_path,['path of video 1','path of video 2'])`
Refine labels (Step 15)	`deeplabcut.refine_labels(config_path)`
Combine datasets (Step 16)	`deeplabcut.merge_datasets(config_path)`

Procedure

▲ **CRITICAL** For full details of how to run each step, please see the code at our GitHub repository: https://github.com/AlexEMG/DeepLabCut; this article is valid as of version 2.0.6. We also provide example Jupyter Notebooks that are executed so the user can see the expected outputs of each step. Additionally, users can run the training and analysis on Colaboratory (a cloud computing platform) in the provided Colab Jupyter Notebook. The following commands will guide the user on how to use the toolbox in ipython.

▲ **CRITICAL** Table 1 is a 'quick-guide' to the minimal commands used for running DeepLabCut in IPython. All these functions have additional optional parameters that the user can change. The user can invoke 'help' for any function and get more information about these optional parameters as follows: in IPython/Jupyter Notebooks: `deeplabcut.nameofthefunction?` in python: `help(deeplabcut.nameofthefunction)`.

Stage I: opening DeepLabCut and creation of a new project ● Timing ~3 min

▲ **CRITICAL** This guide will use the style of Terminal on Ubuntu; if you use Windows, please first install GitBash (https://gitforwindows.org/) and use the program 'cmd'. However, the only major difference is that paths need to be formatted differently; namely, in Windows you must use this notion for paths: r'C:/computername/yourfolder/video1.avi' vs. in Ubuntu/MacOS: '/computername/yourfolder/video1.avi'. For additional details, see the 'Troubleshooting' section for more information.

1 To begin, open the program Terminal. Start an IPython session, and import the package by typing:

```
ipython
import deeplabcut
```

? TROUBLESHOOTING

2 In the Python interpreter, type the following:

```
>> deeplabcut.create_new_project('Name of the project','Name of the
experimenter', ['Full path of video 1','Full path of video2', . . .],
```

```
working_directory='Full path of the working directory',copy_videos=
True/False)
```

The function `create_new_project` creates a new project directory structure and the project configuration file. Each project is identified by the name of the project (e.g., `Reaching`), the name of the experimenter (e.g., `YourName`), as well as the date of creation. Thus, this function requires the user to input the name of the project, the name of the experimenter, and the full path of the videos that are (initially) used to create the training dataset (without spaces in each, i.e., `Test1`, not `Test 1`). Optional arguments specify the working directory, where the project directory will be created, and if the user wants to copy the videos (to the project directory). If the optional argument `working_directory` is unspecified, the project directory is created in the current working directory, and if `copy_videos` is unspecified, symbolic links for the videos are created in the videos directory. Each symbolic link creates a reference to a video and thus eliminates the need to copy the video to the video directory.

To automatically create a variable that points to the `config_path` variable, which will be used below, add:

```
>> config_path = deeplabcut.create_new_project(...
```

You can also create a variable to set this at any point, i.e., config_path = '/home/computername/yourfolder/config.yaml'. Additionally, this set of arguments will create a project directory with the name 'Name of the project+name of the experimenter+date of creation of the project' in the working directory and creates the symbolic links to videos in the 'videos' directory. At this stage, the project directory will have subdirectories: 'dlc-models', 'labeled-data', 'training-datasets', and 'videos' (Fig. 2). Outputs generated during the course of a project will be stored in one of these subdirectories, thus allowing each project to be curated separately from other projects. The purpose of the subdirectories is as follows:

- dlc-models. This directory contains the neural networks with subdirectories 'test' and 'train', each of which holds the meta information with regard to the parameters of the feature detectors in the configuration file. The configuration files are YAML files, a common human-readable data serialization language. These files can be opened and edited with standard text editors. The subdirectory 'train' will store checkpoints (called 'snapshots' in TensorFlow) during training of the model. These snapshots allow the user to reload the trained model without re-training it, or to pick up training from a particular saved checkpoint, in the case that the training was interrupted.
- labeled-data. This directory will store the frames as well as the user annotation. Frames from different videos are stored in separate subdirectories. Each frame has a filename related to the temporal index within the corresponding video, which allows the user to trace each frame back to its origin.
- training-datasets. This directory will contain the training dataset used to train the network and metadata, which contains information about how the training dataset was created.
- videos. This is a directory of video links or videos. When `copy_videos` is set to `False`, this directory contains symbolic links to the videos. If it is set to `True`, then the videos will be copied to this directory. The default is `False`. In addition, if the user wants to add new videos to the project at any stage, the function `add_new_videos` can be used. This will update the list of videos in the project's configuration file. This optional function can be utilized by typing the following:

```
>> deeplabcut.add_new_videos('Full path of the project configuration
file',['full path of video 4', 'full path of video 5'],copy_videos=True/
False)
```

Note that '`Full path of the project configuration file`' will be referenced as a variable called `config_path` throughout this protocol.

▲ **CRITICAL STEP** The project directory also contains the main configuration file called *config.yaml*. The *config.yaml* file contains many important parameters of the project. A complete list of parameters, as well as their descriptions, can be found in Box 1.

? TROUBLESHOOTING

Box 1 | Glossary of parameters in the project configuration file (*config.yaml*)

The *config.yaml* file sets the various parameters for generation of the training set file and evaluation of results. The meaning of these parameters is defined here, as well as referenced in the relevant steps.

Parameters set during the project creation
- `task`: Name of the project (e.g., mouse-reaching). (Set in Step 1; do not edit.)
- `scorer`: Name of the experimenter (set in Step 1; do not edit).
- `date`: Date of creation of the project. (Set in Step 1; do not edit).
- `project_path`: Full path of the project, which is set in Step 1; edit this if you need to move the project to a cluster/server/another computer or a different directory on your computer.
- `video_sets`: A dictionary with the keys as the full path of the video file and the values, `crop` as the cropping parameters used during frame extraction. (Step 1; use the function `add_new_videos` to add more videos to the project; if necessary, the paths can be edited manually, and the `crop` can be edited manually).

Important parameters to edit after project creation
- `bodyparts`: List containing names of the points to be tracked. The default is set to `bodypart1, bodypart2, bodypart3, objectA`. Do not change after labeling frames (and saving labels). You can add additional labels later, if needed.
- `numframes2pick`: This is an integer that specifies the number of frames to be extracted from a video or a segment of video. The default is set to 20.
- `colormap`: This specifies the colormap used for plotting the labels in images or videos in many steps. Matplotlib colormaps are possible (https://matplotlib.org/examples/color/colormaps_reference.html).
- `dotsize`: Specifies the marker size when plotting the labels in images or videos. The default is set to 12.
- `alphavalue`: Specifies the transparency of the plotted labels. The default is set to 0.5.
- `iteration`: This keeps the count of the number of refinement iterations used to create the training dataset. The first iteration starts with 0 and thus the default value is 0. This will auto-increment once you merge a dataset (after the optional refinement stage).

If you are extracting frames from long videos
- `start`: Start point of interval to sample frames from when extracting frames. Value in relative terms of video length, i.e., `[start=0,stop=1]` represents the full video. The default is 0.
- `stop`: Same as `start`, but specifies the end of the interval. Default is 1.

Related to the Neural Network Training
- `TrainingFraction`: This is a two-digit floating-point number in the range [0–1] used to split the dataset into training and testing datasets. The default is 0.95.
- `resnet`: This specifies which pre-trained model to use. The default is 50 (user can choose 50 or 101; see also Mathis et al.[12]).

Used during video analysis (Step 13)
- `batch_size`: This specifies how many frames to process at once during inference (for tuning of this parameter, see Mathis and Warren[27]).
- `snapshotindex`: This specifies which checkpoint to use to evaluate the network. The default is −1. Use `all` to evaluate all the checkpoints. Snapshots refer to the stored TensorFlow configuration, which holds the weights of the feature detectors.
- `pcutoff`: This specifies the threshold of the likelihood and helps to distinguish likely body parts from uncertain ones. The default is 0.1.
- `cropping`: This specifies whether the analysis video needs to be cropped (in Step 13). The default is `False`.
- `x1,x2,y1,y2`: These are the cropping parameters used for cropping novel video(s). The default is set to the frame size of the video.

Used during refinement steps
- `move2corner`: In some (rare) cases, the predictions from DeepLabCut will be outside of the image (because of the location refinement shifts). This binary parameter ensures that those points are mapped to a user-defined point within the image so that the label can be manually moved to the correct location. The default is `True`.
- `corner2move2`: This is the target location, if `move2corner` is `True`. The default is set to `(50,50)`.

Stage II: configuration of the project ● Timing ~5 min

3 Next, open the *config.yaml* file, which was created with `create_new_project`. You can edit this file in any text editor. Familiarize yourself with the meaning of the parameters (Box 1). You can edit various parameters and, in particular, you should add the list of `bodyparts` (or points of interest) that you want to track. For the next data selection step, `numframes2pick` is of particular importance.

▲ **CRITICAL STEP** The 'create a new project' step writes the following parameters to the configuration file: `Task`, `scorer`, `date`, and `project_path`, as well as a list of videos `video_sets`. The first three parameters should not be changed. The list of videos can be changed by the function add _new _videos or manually removing videos. Further parameters are initially added from defaults (Box 1).

Stage III: data selection ● Timing variable, ranging from 10 to 15 min

▲ **CRITICAL** A good training dataset should consist of a sufficient number of frames that capture the full breadth of the behavior. It should reflect the diversity of the behavior with respect to postures,

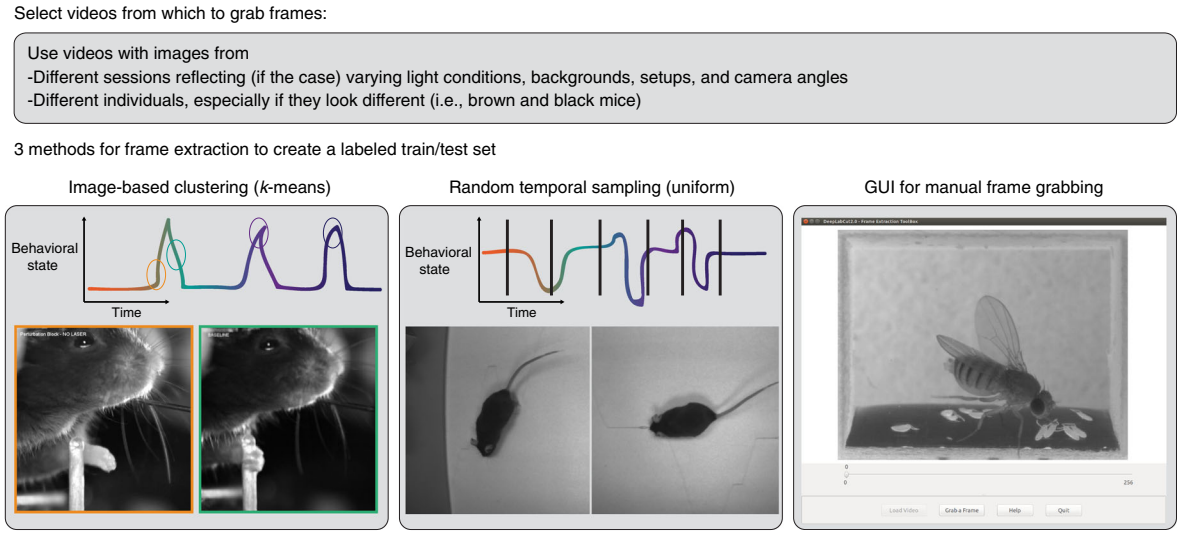

Fig. 3 | Methods for frame selection. The toolbox contains three methods for extracting frames, namely, by clustering based on visual content, by randomly sampling in a uniform way across time, or by manually grabbing frames of interest using a custom GUI. The appropriate method should be selected depending on the studied behavior.

luminance conditions, background conditions, animal identities, and other variable aspects of the data that will be analyzed. Thus, you should select frames from different (behavioral) sessions and different animals if those vary substantially (to train an invariant, robust feature detector). For the behaviors we have tested so far, a dataset of 50–200 frames gave good results[12]. However, depending on the required accuracy and the nature of the scene statistics, more or fewer frames might be necessary to create a sufficient training dataset. Ultimately, to scale up the analysis to large collections of videos with possibly unexpected conditions, one can also refine the dataset in an adaptive way (optional Stage X, Steps 14–16).

4 Use the function `extract_frames` to extract frames from all the videos in the project configuration file in order to create a training dataset. To run this function, type the following:

```
>> deeplabcut.extract_frames(config_path,'automatic/manual','uniform/
kmeans',crop=True/False, userfeedback=True/False)
```

The extracted frames from all the videos are stored in a separate subdirectory named after the video file's name under the 'labeled-data' directory. This function also has various parameters that might be useful depending on the user's need. The default values are `automatic` and `kmeans` clustering, with options to interactively crop the video frames first (crop=True), to use color information for clustering (cluster Color=True), and if you wantto be asked to process a specific video (userfeedback=True).

When running the function `extract_frames`, if the parameter `crop=True`, then frames will be cropped to the interactive user feedback provided (which is then written to the *config.yaml* file). Upon calling `extract_frames`, it will ask the user to draw a boundingbox in the GUI. As a reminder, for each function, place a `?` after the function (i.e., `deeplabcut.extract_frames?`) to see all the available options.

The provided function selects frames from the videos in a temporally uniformly distributed way (`uniform`), by clustering based on visual appearance (`kmeans`), or by `manual` selection (Fig. 3). Uniform selection of frames works best for behaviors in which the postures vary in a temporally independent way across the video. However, some behaviors might be sparse, as in a case of reaching in which the reach and pull are very fast and the mouse is not moving much between trials. In such a case, visual information should be used for selecting different frames If the user chooses to use `kmeans` as a method to cluster the frames, then this function downsamples the video and clusters the frames. Frames from different clusters are then selected. This procedure ensures that the frames look different and is generally preferable. However, on large and long videos, this code is slow due to its computational complexity.

Frame picking is highly dependent on the data and the behavior being studied. Therefore, it is hard to provide all-purpose code that extracts frames to create a good training dataset for every behavior and animal. If users feel that specific frames are lacking, they can extract hand-picked frames of interest using the interactive GUI provided along with the toolbox. This can be launched by using the following (optional) command: `>> deeplabcut.extract_frames(config_path, 'manual')`. The user can use the 'Load Video' button to load one of the videos in the project configuration file, use the scroll bar to navigate across the video, and select 'Grab a Frame' to extract the frame. The user can also look at the extracted frames and, e.g., delete frames (from the directory) that are too similar before manually annotating them. The methods can be used in a mix-and-match way to create a diverse set of frames. The user can also choose to select frames for only specific videos.

▲ **CRITICAL STEP** It is advisable to keep the frame size small, as large frames increase the training and inference times. It is also advisable to extract frames from a period of the video that contains interesting behaviors. This can be achieved by using the `start` and `stop` parameters in the *config.yaml* file. Also, the user can change the number of frames to extract from each video by setting the `numframes2pick` variable in the *config.yaml* file.

? TROUBLESHOOTING

Stage IV: labeling of the frames ● Timing variable, 1–10 h

5 Use the toolbox function `label_frames` to enable easy labeling of all the extracted frames using an interactive GUI. The body parts to label (points of interest) should already have been named in the project's configuration file (config.yaml, in Step 3). The following command invokes the labeling toolbox:

```
>> deeplabcut.label_frames(config_path)
```

6 Next, use the 'Load Frames' button to select the directory that stores the extracted frames from one of the videos. A right click places the first body part, and, subsequently, you can either select one of the radio buttons (top right) to select a body part to label, or use the built-in mechanism that automatically advances to the next body part. If a body part is not visible, simply do not label the part and select the next body part you want to label. Each label will be plotted as a dot in a unique color (see Fig. 4 for more details).

You can also move the label around by left-clicking and dragging. Once the position is satisfactory, you can select another radio button (in the top right) to switch to another label (it also auto-advances, but you can manually skip labels if needed). Once all the visible body parts are labeled, then you can click 'Next' to load the following frame, or 'Previous' to look at and/or adjust the labels on previous frames. You need to save the labels after all the frames from one of the videos are labeled by clicking the 'Save' button. You can save at intermediate points, and then relaunch the GUI to continue labeling (or refine your already-applied labels). Saving the labels will create a labeled dataset in a hierarchical data format (HDF) file and comma-separated (CSV) file in the subdirectory corresponding to the particular video in 'labeled-data'.

▲ **CRITICAL STEP** It is advisable to consistently label similar spots (e.g., on a wrist that is very large, try to label the same location). In general, invisible or occluded points should not be labeled. Simply skip the hidden part by not applying the label anywhere on the frame, or guess the location of body parts, in order to train a neural network that does that as well.

▲ **CRITICAL STEP** Occasionally, the user might want to label additional body parts. In such a case, the user needs to append the new labels to the `bodyparts` list in the *config.yaml* file. Thereafter, the user can call the function `label_frames` and will be asked if he or she wants to display only the new labels or all labels before loading the frames. Saving the labels after all the images are labeled will append the new labels to the existing labeled dataset.

Stage V: (optional) checking of annotated frames ● Timing variable, but typically ~5 min

▲ **CRITICAL** Checking whether the labels were created and stored correctly is beneficial for training, as labeling is one of the most critical parts of a supervised learning algorithm such as DeepLabCut. Nevertheless, this section is optional.

7 The toolbox provides a `check_labels` function. If doing this, use as follows:

```
>> deeplabcut.check_labels(config_path)
```

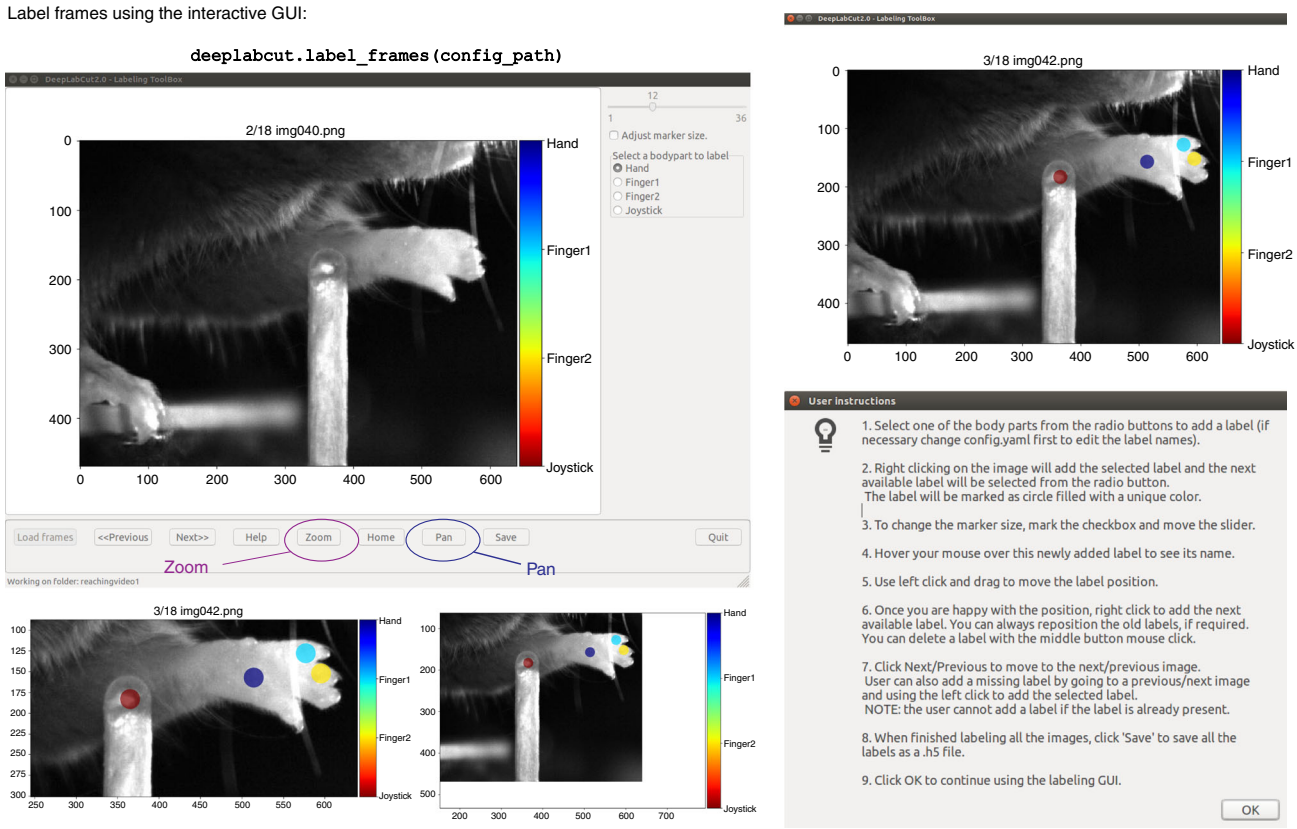

Fig. 4 | Labeling GUI. The toolbox contains a labeling GUI that allows for frame loading, labeling, re-adjustments, and saving the dataset into the correct format for future steps. An example frame is shown, and the help functions are described. Additionally, a user can decide to add more points to an existing labeled dataset by adding the new labels to the *config.yaml* file and re-opening the labeling GUI.

For each directory in 'labeled-data', this function creates a subdirectory with 'labeled' as a suffix. These directories contain the frames plotted with the annotated body parts. You can then double-check whether the body parts are labeled correctly. If they are not correct, use the labeling GUI (Step 5) and adjust the location of the labels.

? TROUBLESHOOTING

Stage VI: creation of a training dataset ● Timing ~1 min

▲ CRITICAL Combining the labeled datasets from all the videos and splitting them will create train and test datasets. The training data will be used to train the network, whereas the test dataset will be used to test the generalization of the network (during evaluation). The function `create_training_dataset` performs these steps.

8 Create a training dataset by typing the following:

```
>> deeplabcut.create_training_dataset(config_path, num_shuffles=1)
```

The set of arguments in the function will shuffle the combined labeled dataset and split it to create a train and a test set. The subdirectory with the suffix 'iteration-#' under the directory 'training-datasets' stores the dataset and meta information, where the '#' is the value of the `iteration` variable stored in the project's configuration file (this number keeps track of how often the dataset is refined; see Stage X). If you wish to benchmark the performance of DeepLabCut, create multiple splits by specifying an integer value in the `num_shuffles` parameter.

Each iteration of the creation of a training dataset will create a .mat file, which contains the address of the images as well as the target postures, and a .pickle file, which contains the meta information about the training dataset. This step also creates a directory for the model, including two subdirectories within 'dlc-models' called 'test' and 'train', each of which has a configuration file called *pose_cfg.yaml*. Specifically, you can edit the *pose_cfg.yaml* within the 'train' subdirectory

Box 2 | Parameters of interest in the network configuration file, *pose_cfg.yaml*

Please note, there are more parameters that typically never need to be adjusted; they can be found in the default *pose_cfg.yaml* file at https://github.com/AlexEMG/DeepLabCut/blob/master/deeplabcut/pose_cfg.yaml.

- `display_iters`: An integer value representing the period with which the loss is displayed (and stored in *log.csv*).
- `save_iters`: An integer value representing the period with which the checkpoints (weights of the network) are saved. Each snapshot has >90 MB, so not too many should be stored.
- `init_weights`: The weights used for training. Default: `<DeepLabCut_path>/Pose_Estimation_Tensorflow/pretrained/resnet_v1_50.ckpt`. For ResNet-50 or 101, -- this will be automatically created. The weights can also be changed to restart from a particular snapshot if training is interrupted, e.g., `<full path>-snapshot-5000` (with no file-type ending added). This would re-start training from the loaded weights (i.e., after 5,000 training iterations, the counter starts from 0).
- `multi_step`: These are the learning rates and number of training iterations to perform at the specified rate. If the users want to stop before 1 million, they can delete a row and/or change the last value to be the desired stop point.
- `max_input_size`: All images larger with size width × height > `max_input_size*max_input_size` are not used in training. The default is 1500 to prevent crashing with an out-of-memory exception for very large images. This will depend on your GPU memory capacity. However, we suggest reducing the pixel size as much as possible; see Mathis and Warren[27].

The following parameters allow one to change the resolution:

- `global_scale`: All images in the dataset will be rescaled by the following scaling factor to be processed by the convolutional neural network. You can select the optimal scale by cross-validation (see discussion in Mathis et al.[12]). Default is 0.8.
- `pos_dist_thresh`: All locations within this distance threshold (measured in pixels) are considered positive training samples for detection (see discussion in Mathis et al.[12]). Default is 17.

The following parameters modulate the data augmentation. During training, each image will be randomly rescaled within the range `[scale_jitter_lo, scale_jitter_up]` to augment training:

- `scale_jitter_lo`: 0.5 (default).
- `scale_jitter_up`: 1.5 (default).
- `mirror`: If the training dataset is symmetric around the vertical axis, this Boolean variable allows random augmentation. Default is `False`.
- `cropping`: Allows automatic cropping of images during training. Default is `True`.
- `cropratio`: Fraction of training samples that are cropped. Default is 40%.
- `minsize`, `leftwidth`, `rightwidth`, `bottomheight`, `topheight`: These define dimensions and limits for auto-cropping.

before starting the training. These configuration files contain meta information to configure feature detectors, as well as the training procedure. Typically, these do not need to be edited for most applications. Key parameters are defined in Box 2.

Stage VII: training the network ● Timing variable, ranging from 1 to 12 h

▲ **CRITICAL** It is recommended to train for thousands of iterations (typically >100,000) until the loss plateaus. The variables `display_iters` and `save_iters` in the *pose_cfg.yaml* file allow the user to alter how often the loss is displayed and how often the (intermediate) weights are stored.

? TROUBLESHOOTING

9 Use the function `train_network` to train the network by typing the following:

```
>> deeplabcut.train_network(config_path)
```

The set of arguments in the function starts training the network for the dataset created for one specific shuffle. Example parameters that one can call are given below:

```
train_network(config_path,shuffle=1,trainingsetindex=0,gputouse=
None,max_snapshots_to_keep=5,   displayiters=1000,saveiters=20000,
maxiters=200000)
```

Important parameters are as follows:

- `config_path`. Full path of the *config.yaml* file as a string (or a variable that points to the path).
- `shuffle`. Integer value specifying the shuffle index for the model to be trained. The default is set to 1.
- `trainingsetindex`. Integer specifying which training set fraction to use. By default it is the first index value listed (note that `Training Fraction` is a list in *config.yaml*).
- `gputouse`. An integer indicating the number of your GPU (see the number in nvidia-smi; installed with GPU support above, but also see: https://developer.nvidia.com/nvidia-system-management-interface). If there is only one GPU, it is typically automatically engaged without any change. See also: https://nvidia.custhelp.com/app/answers/detail/a_id/3751/~/useful-nvidia-smi-queries.

Additional parameters are as follows:

- `displayiters`. This sets how often the network iterations are displayed in the terminal. This variable is set in *pose_config.yaml* (called `display_iters`). However, you can overwrite it by passing a variable. If `None`, the value from *pose_config.yaml* is used; otherwise, it is overwritten.
- `saveiters`. This sets how many iterations to save; every 50,000 is the default. This variable is set in *pose_config.yaml* (called `save_iters`). However, you can overwrite it by passing a variable. If `None`, the value from the *pose_config.yaml* file is used; otherwise, it is overwritten.
- `maxiters`. This variable sets how many iterations to train for. This variable is set in *pose_config.yaml*. However, you can overwrite it by passing a variable. If `None`, the value from there is used; otherwise it is overwritten. Default: `None`.
- `max_snapshots_to_keep`. This sets how many snapshots are kept, i.e., states of the trained network. For each saving iteration, a snapshot is stored; however, only the last `max_snapshots_to_keep` are kept. If you change this to `None`, then all are kept.

By default, the pre-trained ResNet networks are not downloaded with the DeepLabCut toolbox (as it has a size of ~100 MB). However, if not previously downloaded from the TensorFlow model weights, it will be downloaded to a subdirectory 'pre-trained' under the subdirectory 'models' in 'pose_estimation_tensorflow', where DeepLabCut is installed (i.e., usually in Python's site-packages). During training, checkpoints are stored in the subdirectory 'train' under the respective iteration directory at user-specified iterations ('save_iters'). If you wish to restart the training from a specific checkpoint, specify the full path of the checkpoint for the variable `init_weights` in the *pose_cfg.yaml* file under the 'train' subdirectory before starting to train (Box 2).

Stage VIII: evaluation of the trained network ● Timing variable, typically 2–10 min

▲ CRITICAL It is important to evaluate the performance of the trained network. This performance is measured by computing the mean average Euclidean error (MAE; which is proportional to the average root mean square error) between the manual labels and the ones predicted by DeepLabCut. The MAE is saved as a comma-separated file and displayed for all pairs and only likely pairs (`>p-cutoff`). This helps to exclude, for example, occluded body parts. One of the strengths of DeepLabCut is that, owing to the probabilistic output of the scoremap, it can, if sufficiently trained, also reliably report whether a body part is visible in a given frame (see discussions of fingertips in reaching and the *Drosophila* legs during 3D behavior in Mathis et al.[12]).

10 Evaluate the network by typing the following:

```
>> deeplabcut.evaluate_network(config_path, plotting=True)
```

Setting `plotting` to `True` plots all the testing and training frames with the manual and predicted labels. You should visually check the labeled test (and training) images that are created in the 'evaluation-results' directory. Ideally, DeepLabCut labeled the unseen (test) images according to your required accuracy, and the average train and test errors will be comparable (good generalization). What (numerically) constitutes an acceptable MAE depends on many factors (including the size of the tracked body parts and the labeling variability). Note that the test error can also be larger than the training error because of human variability in labeling (see Fig. 2 in Mathis et al.[12]).

If desired, customize the plots by editing the *config.yaml* file (i.e., the colormap, marker size (dotsize), and transparency of labels (alphavalue) can be modified). By default, each body part is plotted in a different color (governed by the colormap) and the plot labels indicate their source. Note that, by default, the human labels are plotted as a plus symbol ('+') and DeepLabCut's predictions are plotted either as a dot (for confident predictions with likelihood `>p-cutoff`) or as an 'x' (for likelihood `≤p-cutoff`). Example test and training plots from various projects are depicted in Fig. 5.

The evaluation results for each shuffle of the training dataset are stored in a unique subdirectory in a newly created 'evaluation-results' directory in the project directory. You can visually inspect whether the distance between the labeled and the predicted body parts is acceptable. In the event of benchmarking with different shuffles of the same training dataset, you can provide multiple shuffle indices to evaluate the corresponding network. If the generalization is not sufficient, you might want to:

- Check if the body parts were labeled correctly, i.e., invisible points are not labeled and the points of interest are labeled accurately (Step 7);

Fig. 5 | Evaluation of results. a, The code that is used to evaluate the network and the output the user will see in the terminal are displayed for one example. **b**,**c**, Another critical output of the evaluation steps are labeled frames from training (**b**) and test (**c**) images, as shown for a cheetah project. Note that the human labels are plotted as a plus symbol ('+') and DeepLabCut's predictions are plotted either as a dot (for confident predictions with likelihood >`p-cutoff`) or as an 'x' (for likelihood ≤`p-cutoff`). **d**, Additional example test evaluation images from *Drosophila* (different images from the network trained in Mathis et al.[12]) and horse-tracking projects.

- Make sure that the loss has already converged (Step 9; i.e., make sure the loss displayed has plateaued);
- Change the augmentation parameters (see Box 2);
- Consider labeling additional images and make another iteration of the training dataset (optional Stage X).

? TROUBLESHOOTING

Stage IX: video analysis and plotting of results ● Timing variable

▲ CRITICAL The trained network can be used to analyze new videos. The user needs to first choose a checkpoint with the best evaluation results for analyzing the videos. In this case, the user can specify the corresponding index of the checkpoint in the variable `snapshotindex` in the *config.yaml* file. By default, the most recent checkpoint (i.e., last) is used for analyzing the video.

11 Analyze a video (or a folder of videos) by typing the following:

```
>> deeplabcut.analyze_videos(config_path,['Full path of video or video folder'], shuffle=1, save_as_csv=True, videotype='.avi')
```

The labels are stored in a multi-index Pandas array[43], which contains the name of the network, body part name, (*x*, *y*) label position in pixels, and the likelihood for each frame per body part. These arrays are stored in an efficient HDF format in the same directory where the video is stored. However, if the flag `save_as_csv` is set to `True`, the data can also be exported in CSV format files, which in turn can be imported into many programs, such as MATLAB, R, and Prism; this flag is set to `False` by default. Instead of the video path, one can also pass a directory, in which case all videos of the type 'videotype' in that folder will be analyzed. For some projects, time-lapsed images are taken, for which each frame is saved independently. Such data can be analyzed with the function `deeplabcut.analyze_time_lapse_frames`.

? TROUBLESHOOTING

12 The labels for each body part across the video ('trajectories') can also be filtered and plotted after `analyze_videos` is run (which has many additional options (seedeeplabcut.filterpredictions?)). We also provide a function to plot the data overtime and pixels in frames. The provided plotting

function in this toolbox utilizes matplotlib[44]; therefore, these plots can easily be customized. To call this function, type the following:

```
>> deeplabcut.filterpredictions(config_path,['video_path'], videotype='.avi')
>> deeplabcut.plot_trajectories(config_path,['Full path of video'])
```

The ouput files can also be easily imported into many programs for further behavioral analysis (see Stage XI and 'Anticipated results').

13 In addition, the toolbox provides a function to create labeled videos based on the extracted poses by plotting the labels on top of the frame and creating a video. To use it to create multiple labeled videos (provided either as each video path or as a folder path), type the following:

```
>> deeplabcut.create_labeled_video(config_path,['Full path of video 1', 'Full path of video 2'])
```

This function has various parameters; in particular, the user can set the `colormap`, the `dotsize`, and the `alphavalue` of the labels in the *config.yaml* file, and can pass a variable called `displayedbodyparts` to select only a subset of parts to be plotted. The user can also save individual frames in a temp-directory by passing `save_frames=True` (this also creates a higher-quality video). All parameters are listed in the related help function (`deeplabcut.create_labeled_video?`).

Stage X: (optional) network refinement—extraction of outlier frames ● Timing variable

14 Although DeepLabCut typically generalizes well across datasets, one might want to optimize its performance in various, perhaps unexpected, situations. For generalization to large datasets, images with insufficient labeling performance can be extracted and manually corrected by adjusting the labels to increase the training set and iteratively improve the feature detectors. Such an active learning framework can be used to achieve a predefined level of confidence for all images with minimal labeling cost (discussed in Mathis et al.[12]). Then, owing to the large capacity of the neural network that underlies the feature detectors, one can continue training the network with these additional examples. One does not necessarily need to correct all errors, as common errors can be eliminated by relabeling a few examples and then retraining. A priori, given that there is no ground truth data for analyzed videos, it is challenging to find putative 'outlier frames'. However, one can use heuristics such as the continuity of body part trajectories to identify images where the decoder might make large errors. We provide various frame-selection methods for this purpose. In particular, the user can do the following:

- Select frames if the likelihood of a particular or all body parts lies below p_{bound} (note this could also be due to occlusions rather than errors);
- Select frames in which a particular body part or all body parts jumped more than epsilon pixels from the last frame;
- Select frames if the predicted body part location deviates from a state-space model[45,46] fit to the time series of individual body parts. Specifically, this method fits an AutoRegressive Integrated Moving Average (ARIMA) model to the time series for each body part. Thereby, each body part detection with a likelihood smaller than p_{bound} is treated as missing data. An example fit for one body part can be found in Fig. 6a. Putative outlier frames are then identified as time points, at which the average body part estimates are at least epsilon pixels away from the fits. The parameters of this method are epsilon, p_{bound}, the ARIMA parameters, and the list of body parts to consider (can also be 'all').

Refine a specific video by typing the following:

```
>> deeplabcut.extract_outlier_frames(config_path,['videofile_path'])
```

This step has many parameters that can be set. Type `deeplabcut.extract_outlier_frames?` to see the full range of possible parameters.

In general, depending on the parameters, these methods might return many more frames than the user wants to extract (`numframes2pick`). Thus, this list is then used to select outlier frames

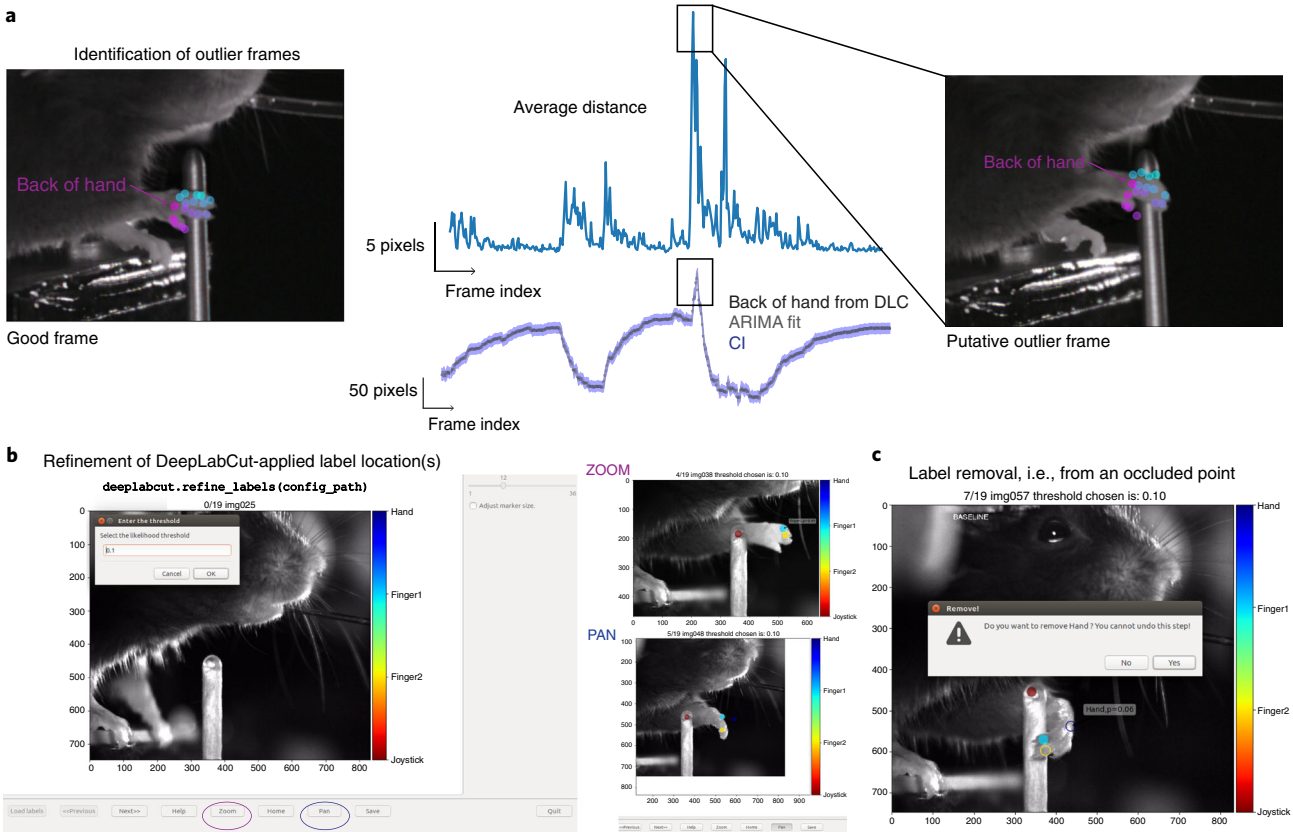

Fig. 6 | Refinement tools. A user can refine the network by first extracting outliers and then by manually correcting the annotations through a dedicated graphical user interface. Three methods are available; here, we illustrate the state-space model fit. **a**, Outlier detection: for illustration, we depict the *x* coordinate estimate by DeepLabCut (DLC) for the back of the hand and an ARIMA fit with 99% confidence interval (CI, bottom). The average Euclidean distance for the tracked 17 parts of the hand to the fit is also depicted (in blue, top). This distance can be used to find putative outliers as indicated by the corresponding frames. **b**,**c**, The labels can be adjusted (**b**) and occluded labels can be removed (**c**).

either by randomly sampling from this list ('uniform') or by performing '*k*-means' clustering on the corresponding frames (same methodology and parameters as in Step 4). Furthermore, before this second selection happens, you are informed about the number of frames satisfying the criteria and asked if the selection should proceed. This step allows you to perhaps change the parameters of the frame-selection heuristics first. The `extract_outlier_frames` command can be run iteratively and can (even) extract additional frames from the same video. Once enough outlier frames are extracted, use the refinement GUI to adjust the labels based on any user feedback (Step 15, below).

Stage X continued: (optional) refinement of labels—augmentation of the training dataset

● **Timing variable**

15 Based on the performance of DeepLabCut, four scenarios are possible:

- *A visible body part with an accurate DeepLabCut prediction.* These labels do not need any modification.
- *A visible body part, but the wrong DeepLabCut prediction.* Move the label's location to the actual position of the body part.
- *An invisible, occluded body part.* Remove the predicted label by DeepLabCut with a right click. Every predicted label is shown, even when DeepLabCut is uncertain. This is necessary, so that the user can potentially move the predicted label. However, to help the user to remove all invisible body parts, the low-likelihood predictions are shown as open circles (rather than disks).
- *Invalid image.* in the unlikely event that there are any invalid images, the user should remove such images and their corresponding predictions, if any. Here, the GUI will prompt the user to remove an image identified as invalid.

Refine the labels for extracted putative outlier frames by typing the following to open the GUI:

```
>> deeplabcut.refine_labels(config_path)
```

This will launch a GUI with which you can refine the labels (Fig. 6). Use the 'Load Labels' button to select one of the subdirectories where the extracted frames are stored. Each label will be identified by a unique color. To identify low-confidence labels, specify the threshold of the likelihood. This causes the body parts with likelihood below this threshold to appear as circles and the ones above the threshold to appear as solid disks while retaining the same color scheme. Next, to adjust the position of the label, hover the mouse over the label to identify the specific body part, then left-click it and drag it to a different location. To delete a label, right-click on the label (once a label is deleted, it cannot be retrieved).

16 After correcting the labels for all the frames in each of the subdirectories, merge the dataset to create a new dataset. To do this, type the following:

```
>> deeplabcut.merge_datasets(config_path)
```

The `iteration` parameter in the *config.yaml* file will be automatically updated.

Once the datasets are merged, you can test if the merging process was successful by plotting all the labels (Step 7). Next, with this expanded image set, you can now create a novel training set and train the network as described in Steps 8 and 9. The training dataset will be stored in the same place as before but under a different 'iteration- #' subdirectory, where the '#' is the new value of `iteration` variable stored in the project's configuration file (this is automatically done).

If, after training, the network generalizes well to the data (i.e., run `evaluate_network` in Step 10), proceed to Step 11 to analyze new videos. Otherwise, consider labeling more data (optional Stage X).

Stage XI: working with the output files of DeepLabCut ● Timing variable

17 Once you have a trained network with your desired level of performance (i.e., Steps 1–10 with/without optional refinement) and have used DeepLabCut to analyze videos (i.e., used the trained network snapshot weights to analyze novel videos by running Step 11), you will have files that contain the predicted *x* and *y* pixel coordinates of each body part and the network confidence likelihood (per frame). These files can then be loaded into many programs for further analysis. For example, you can compute general movement statistics (e.g., position, velocity) or interactions, e.g., with (tracked) objects in the environment. The output from DeepLabCut can also interface with behavioral clustering tools such as JAABA[47], MotionMapper[48], an AR-HMM[49,50], or other clustering approaches such as iterative denoising tree (IDT) methods[51,52]; for a review, see Todd et al.[53]. In addition, users are contributing analysis code for the outputs of DeepLabCut here: https://github.com/AlexEMG/DLCutils.

Beyond extracting poses from videos (Step 11), there are options to generate several plots (by running Step 12) and to generate a labeled video (by running Step 13). The following list summarizes the main result files and directories generated:

- Step 9 creates network weights, which are stored as *snapshot-#.meta*, *snapshot-#.index* and *snapshot-#.data-00000-of-00001*; these TensorFlow files contain the weights and can be used to analyze videos (Step 13). If the training was interrupted, it can be restarted from a particular snapshot (Box 2). These files are periodically saved in a subdirectory ('train') in the 'dlc-models' directory. The '#' in the filename specifies the training iteration index.
- Step 11 generates predicted DeepLabCut labels, which are stored as .HDF and/or .CSV files named *<name_of_new_video>DeepCut_resnet<#>_<Task><date>shuffle<num_shuffles>_<snapshotindex>.h5*; this file is saved in the same location where the video is located. This file contains the name of the network, body part names, *x*- and *y*-label positions in pixels, and the prediction likelihood for each body part per frame of the video. The filename is based on the name of the new video (*<name_of_new_video>*), the number of ResNet layers used (*<#>*), the task and date of the experiment (*<Task>* and *<date>*, respectively), shuffle index of the training dataset (*<num_shuffle>*), and the training snapshot file index (*<snapshotindex*) used for making the predictions.

- Step 12 generates several plots for an analyzed video. It creates a folder called 'plot-poses' (in the directory of the video). The plots display the coordinates of body parts versus time, likelihoods versus time, the x versus y coordinates of the body parts, and histograms of consecutive coordinate differences. These plots help the user to quickly assess the tracking performance for a video. Ideally, the likelihood stays high and the histogram of consecutive coordinate differences has values close to zero (i.e., no jumps in body-part detections across frames).
- Step 13 creates a labeled video that will be named *<name_of_new_video>DeepCut_resnet<#>_<Task><date>shuffle<num_shuffle>_<snapshotindex>_labeled.mp4;* This file is saved in the same location as that of the original video. The properties of the labels (color, dotsize, and others) can be changed in the *config.yaml* file (Step 3), and video compression can be used.

Troubleshooting

The Jupyter Notebooks provided on GitHub have been executed already so the user can see the expected outputs. They are also annotated with potential errors and considerations. In addition, when running the Notebooks on Colaboratory (https://colab.research.google.com), users can directly search for any error messages that arise, as they can involve TensorFlow or other packages not managed by us. In the DeepLabCut code, we strived to make error messages intuitive for each step, and each output directs the user to the next step in the process.

There is also an actively maintained GitHub repository that has an 'Issues' section in which users can look for solutions to problems others have encountered or report new ones (https://github.com/AlexEMG/DeepLabCut/issues), as well as a wiki page (https://github.com/AlexEMG/DeepLabCut/wiki/Troubleshooting-Tips).

Troubleshooting advice can be found in Table 2.

Table 2 | Troubleshooting table

Step	Problem	Possible reason	Solution
1	DeepLabCut module is not found	If you installed it in an Anaconda environment, you need to activate the environment before you start IPython and import DeepLabCut	Activate the environment. Windows: `activate <name of environment>` Linux/Mac: `source activate <name of environment>`
1,2,4,7,8,10,11	You do not have sufficient privilege to perform this action	This error is specific to Windows users. Users need administrative rights to write the data	Open 'cmd' as an administrator before importing DeepLabCut, and then retry the step

8	FileNotFoundError (or ValueError): The passed save_path is not a valid checkpoint: xxx/Deeplabcut/lib/site-packages/deeplabcut/pose_estimation_tensorflow/models/pretrained/resnet_v1_50.ckpt	The ResNet weights were not found. This is specific to Windows and some Docker users and is related to insufficient permissions	First, change the current working directory to the directory where deeplabcut is installed: e.g.,
			`cd /xxx/xxx/anaconda3/envs/<name of environment>/lib/python3.6/site-packages/deeplabcut/pose_estimation_tensorflow/models/pretrained/`
			Then run
			`sudo bash download.sh`
			Then change the permissions:
			`sudo chown yourusername: yourusername resnet_v1_50.ckpt`
			Also see https://github.com/AlexEMG/DeepLabCut/wiki/Troubleshooting-Tips

11	SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated /UXXXXXXXX escape	This error is due to an incorrect way of passing the videos for analysis. It is specific to Windows users	Either prefix the command with 'r' or duplicate all backslashes, e.g.,
			`rC:/Users/computername/Videos/reachingvideo1.avi`
			or
			`C://Users//computername//Videos//reachingvideo1.avi`

Timing

The times listed below are for a Dell 7920 (i9 CPU, 32 GB of RAM), running an Ubuntu 16.04 LTS system, using a 1080-Ti GPU, but CPU times are noted where appropriate. Timing also includes time to type commands into the terminal and to edit the .yaml file(s) in a word processor, in addition to the network computation times.

Installation time is highly dependent on the manner chosen, i.e., users can use the fully installed Docker container that is provided, use the provided conda environments, or make their own. We anticipate it will take a user 10–60 min for installation, depending on the method chosen.

The time required to select the data in Step 4 is mainly related to the mode of selection, the number of videos, and the algorithm used to extract the frames. It takes 30 s/video in an automated mode with default parameter settings (processing time strongly depends on length and size of the video). However, it may take longer in the manual mode and mainly depends on the time it takes to find frames with diverse behavior.

The time required to label the frames (Stage IV) depends on the speed of the experimenter in identifying the correct label and is mainly related to the number of body parts and the total number frames.

The time required to train the network (Stage VII) mainly depends on the frame size of the dataset and the computer hardware. On an NVIDIA GeForce GTX 1080 Ti GPU, it takes 4 h to train the network for 200,000 iterations (640 × 480 pixels). On a CPU, it will take several days to train for the same number of iterations on the same dataset.

The time required to evaluate the trained network (Stage VIII) depends on the computational hardware, the number of checkpoints to evaluate, and the number of images in the training dataset. With the default parameters and ~150 images in the training dataset, it will take ≈30 s/checkpoint on a GPU. However, for the same snapshot and training set, a CPU takes 10–100 times longer.

The time required to analyze a video (Stage IX) is mainly related to the number of frames in the video and whether the computational power of a GPU is available. With a GPU, analysis runs at 10–90 FPS depending on the frame size (1,000 × 1,000 pixels to 200 × 200 pixels, respectively), with a batch size of 1. The analysis speed can be markedly improved by increasing the batch size in the config file (i.e., to 32 or 64; see Mathis and Warren[27]).

The time required to select the data (Stage X) is mainly related to the mode of selection, number of videos, and the algorithm used to extract the frames. It takes ~1 min/video in an automated mode with default parameter settings (processing time strongly depends on the length and size of video).

The time required to refine the labels (Stage X) depends on the speed of the experimenter in identifying incorrect labels and is mainly related to the number of body parts and the total number of extracted outlier frames.

Protocol timing

Steps 1 and 2, Stage I, starting Python and creation of a new project: ~3 min
Step 3, Stage II, configuration of the *config.yaml* file: ~5 min
Step 4, Stage III, extraction of frames to label: variable, ranging from 10 to 15 min
Steps 5 and 6, Stage IV, manual labeling of frames: variable, ranging from 1 to 10 h
Step 7, Stage V, checking labels: 5 min, depending on how many frames were labeled
Step 8, Stage VI, creation of training set: ~1 min
Step 9, Stage VII, training the network: ~1–12 h (~6 h on a 1080-Ti GPU, and dependent on frame size)
Step 10, Stage VIII, evaluation of the network: variable, but typically 2–10 min
Steps 11–13, Stage IX, novel video analysis: variable, (10–1,000 FPS); i.e., a 20-min video collected at 100 FPS, frame size 138 × 138, and batch size set to 64 will take ~3.5 min on a GPU
Steps 14–16, (optional) Stage X, refinement: timing is equivalent timing to that for Steps 4–10
Step 17, Stage XI, working with the output files of DeepLabCut: variable

Anticipated results

DeepLabCut has been used for pose estimation of various points of interest for different behaviors and animals (Fig. 1). We showed in our original paper describing this software[12] that because of the pre-training of the ResNets on ImageNet[54], DeepLabCut does not require many labeled frames. We previously showed that as few as 50 frames could be labeled for accurate tracking of a mouse snout in an open-field-like scenario with challenging background and lighting statistics. The technical report

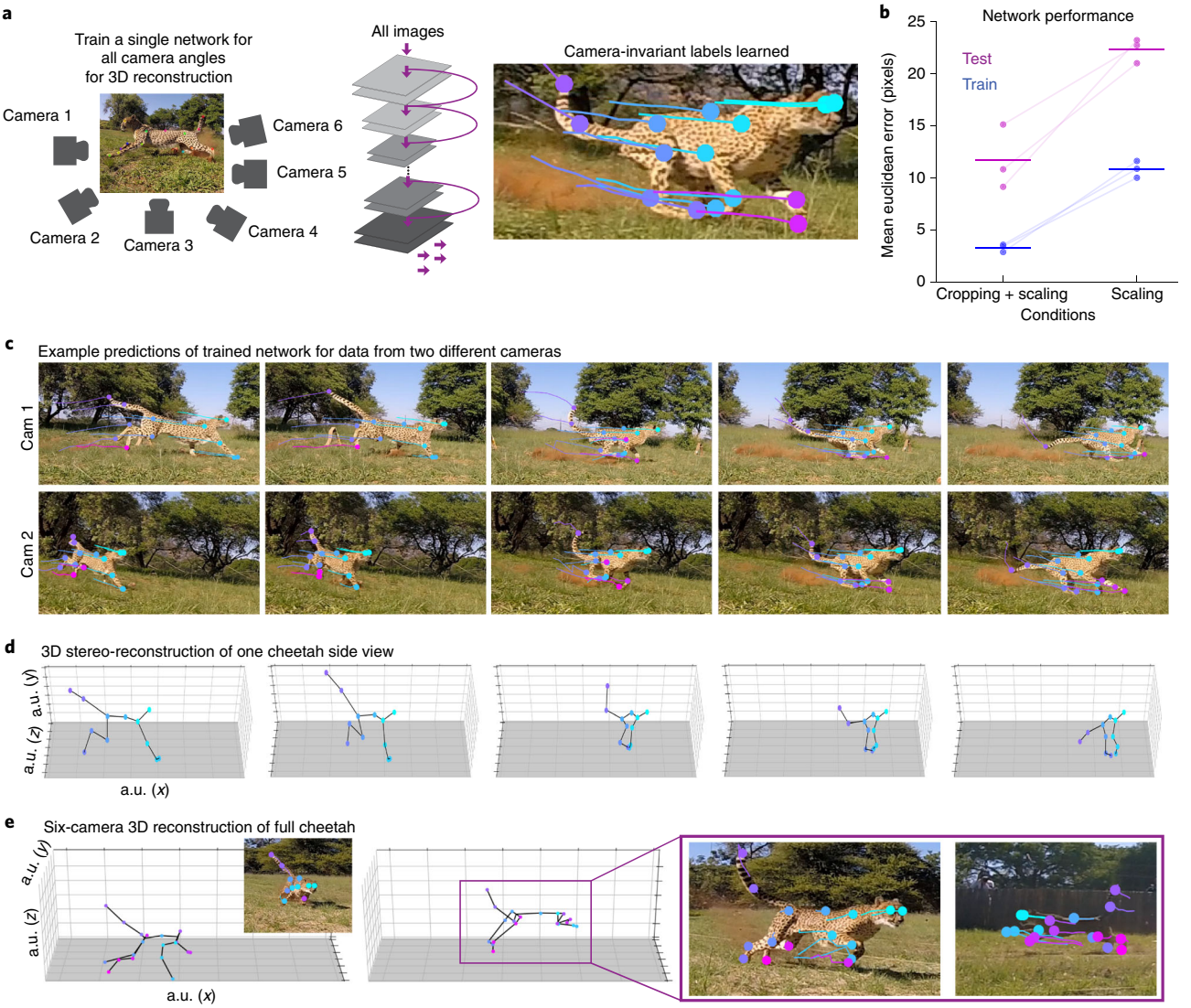

Fig. 7 | 3D pose estimation of a cheetah. An example use case for pose estimation of a hunting cheetah. **a**, Images collected from multiple cameras can be used to train a single network. **b**, Cross-validation: training and test performance after training for 300,000 iterations by augmentation with scaling only and auto-cropping plus scaling (new feature) for a dataset comprising 756 labeled images for three splits with 95% training set fraction. Automatic cropping substantially improves training and test performance for this dataset composed of very large images (average test error: 11.7 ± 1.5 pixels vs. scaling only: 22.3 ± 0.5, n=3 shuffles). **c**, Example frames with overlaid body part estimates extracted by DeepLabCut, as well as a trajectory of past estimates from previous frames. The same network was trained for all camera views, and is used to extract the poses (256 labeled images, 99% training set split). **d**, 3D reconstruction of body parts from cheetah's side facing both cameras from the same series of frames as in **b**. **e**, 3D reconstruction of the full body of the cheetah (for a different cheetah) using six cameras. The right panels show two example camera views that were used to create the 3D skeleton.

also contains examples for mouse hand articulation and fruit fly pose estimation during egg laying[12]. It has also been used for human babies[26], hand-gesture communication in humans[55], 3D-locomotion studies in rodents[27], tracking of rodents equipped with miniature neural stimulators[56], and multi-body-part tracking—including eye-tracking—during perceptual decision making[28]. Other (currently unpublished) use cases can be found at www.deeplabcut.org.

Here, we provide a novel use case in which collaborators wanted to track the 3D kinematics of a cheetah performing a high-speed pursuit task as occurs when hunting in the wild[57]. This specific application has many challenges for any pose-estimation approach: highly variable scene statistics where the animal traverses over a field of grass and stirs up dust, different weather conditions (as the experiments were carried out over multiple months), the size of the animal in the frame changing dramatically as it runs into a 'camera trap' of six high-speed cameras (90 FPS, 1,080-pixel resolution), different cheetahs that have diverse coats that blend well with background foliage, and the cheetahs'

highly articulated leg and body movements (Fig. 5a,b). In total, we iteratively labeled ~900 frames across multiple cameras using the toolbox presented in this guide. Thus, the 3D kinematics can be reconstructed from one network being trained on multiple views, and the user needs only the camera calibration files to reconstruct the data. This camera calibration and triangulation can be done in many programs, including the free package OpenCV[37] (and see custom functions at https://github.com/AlexEMG/DeepLabCut).

In Fig. 7, we show how multiple cameras can be used to train a single network (Fig. 7a). Specifically, we first selected 10 frames each from 12 different videos by using uniform random sampling across the length of each video. Next, labeling was performed on the initial set using the supplied GUI. We trained with automatic cropping and re-scaling; owing to the large input image size of 1,920 × 1,080 pixels, we changed `max_input_size` to 2,000. After training, we iteratively refined the images and re-trained the network until we had a set of 256 frames that allowed for reasonable performance for some sessions (Fig. 7a,c,d). We also added a novel augmentation feature to this toolbox, which we benchmarked on a larger dataset of 756 labeled frames (Fig. 7b,e); it substantially improved the performance (Fig. 7b). In Fig. 7c, we show an example image sequence with trajectories of past actions (also see Supplementary Video 1), and using two cameras, we show a 3D reconstruction[58,59] of one body side (Fig. 7d); we also show a full six-camera reconstruction (Fig. 7e). We used the easyWand toolbox[60] to calculate the required sparse bundle adjustment calibration coefficients for reconstruction and applied these to the DeepLabCut output files (per camera). For the experiment in Fig. 7e, the standard deviation was 1.73 mm (Wand score of 3.1375%), and the mean reprojection error was 1.28 pixels (for the six cameras). Then, to plot the data, we applied a likelihood threshold of 0.5 for a point to be included in the 3D image. This is an ongoing project studying the biomechanics of cheetah hunting, and, hence, further details will be published elsewhere. Yet we hope this example use case shows how DeepLabCut can be used even for very challenging, high-resolution video tracking problems in the wild and will empower users to apply this approach to their own unique and exciting experiments.

Reporting Summary

Further information on research design is available in the Nature Research Reporting Summary linked to this article.

Data and code availability

The code is fully available at https://github.com/AlexEMG/DeepLabCut. Other inquiries should be made to the corresponding author (M.W.M.).

References

1. Ilg, E. et al. FlowNet 2.0: evolution of optical flow estimation with deep networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 1647–1655 (2017).
2. Toshev, A. & Szegedy, C. DeepPose: human pose estimation via deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVVR)* 1653–1660 (2014).
3. Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. & Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **37**, 421–436 (2018).
4. Wainberg, M., Merico, D., Delong, A. & Frey, B. J. Deep learning in biomedicine. *Nat. Biotechnol.* **36**, 829–838 (2018).
5. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
6. Donahue, J. et al. DeCAF: a deep convolutional activation feature for generic visual recognition. *Proceedings of the 31st International Conference on Machine Learning* 647–655 (2014).
7. Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)* **27**, 3320–3328 (2014).
8. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* 1 (MIT Press, Cambridge, MA, 2016).
9. Kümmerer, M., Wallis, T. S. A., Gatys, L. A. & Bethge, M. Understanding low-and high-level contributions to fixation prediction. *Proceedings of the IEEE International Conference on Computer Vision* 4789–4798 (2017).
10. Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M. & Schiele, B. DeeperCut: a deeper, stronger, and faster multi-person pose estimation model. *European Conference on Computer Vision* 34–50 (2016).
11. Insafutdinov, E. et al. ArtTrack: articulated multi-person tracking in the wild. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 1293–1301 (2017).
12. Mathis, A. et al. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nat. Neurosci.* **21**, 1281–1289 (2018).
13. Nath, T. et al. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Preprint at https://www.biorxiv.org/content/10.1101/476531v1 (2018).

14. Dell, A. I. et al. Automated image-based tracking and its application in ecology. *Trends Ecol. Evol. (Amst)* **29**, 417–428 (2014).
15. Anderson, D. J. & Perona, P. Toward a science of computational ethology. *Neuron* **84**, 18–31 (2014).
16. Egnor, S. R. & Branson, K. Computational analysis of behavior. *Annu. Rev. Neurosci.* **39**, 217–236 (2016).
17. Dollár, P., Welinder, P. & Perona, P. Cascaded pose regression. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 1078–1085 (2010).
18. Gomez-Marin, A., Partoune, N., Stephens, G. J. & Louis, M. Automated tracking of animal posture and movement during exploration and sensory orientation behaviors. *PLoS ONE* **7**, 1–9 (2012).
19. Matsumoto, J. et al. A 3D-video-based computerized analysis of social and sexual interactions in rats. *PLoS ONE* **8**, e78460 (2013).
20. Uhlmann, V., Ramdya, P., Delgado-Gonzalo, R., Benton, R. & Unser, M. FlyLimbTracker: an active contour based approach for leg segment tracking in unmarked, freely behaving *Drosophila*. *PLoS ONE* **12**, e0173433 (2017).
21. Ben-Shaul, Y. OptiMouse: a comprehensive open source program for reliable detection and analysis of mouse body and nose positions. *BMC Biol.* **15**, 41 (2017).
22. Winter, D. A. *Biomechanics and Motor Control of Human Movement* (John Wiley & Sons, 2009).
23. Zhou, H. & Hu, H. Human motion tracking for rehabilitation—a survey. *Biomed. Signal Process. Control* **3**, 1–18 (2008).
24. Kays, R., Crofoot, M. C., Jetz, W. & Wikelski, M. Terrestrial animal tracking as an eye on life and planet. *Science* **348**, aaa2478 (2015).
25. Colyer, S. L., Evans, M., Cosker, D. P. & Salo, A. I. A review of the evolution of vision-based motion analysis and the integration of advanced computer vision methods towards developing a markerless system. *Sports Med. Open* **4**, 24 (2018).
26. Wei, K. & Kording, K. P. Behavioral tracking gets real. *Nat. Neurosci.* **21**, 1146–1147 (2018).
27. Mathis, A. & Warren, R. A. On the inference speed and video-compression robustness of DeepLabCut. Preprint at https://www.biorxiv.org/content/10.1101/457242v1 (2018).
28. Aguillon Rodriguez, V. et al. The International Brain Laboratory: reproducing a single decision-making behavior in mice across labs. Society for Neuroscience 2018, abstr. 613.01 (2018).
29. Felzenszwalb, P. F. & Huttenlocher, D. P. Pictorial structures for object recognition. *Int. J. Comput. Vis* **61**, 55–79 (2005).
30. Andriluka, M., Pishchulin, L., Gehler, P. & Schiele, B. 2D human pose estimation: new benchmark and state of the art analysis. *2014 IEEE Conference on Computer Vision and Pattern Recognition* 3686–3693 (2014).
31. Wei, S.-E., Ramakrishna, V., Kanade, T. & Sheikh, Y. Convolutional pose machines. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 4724–4732 (2016).
32. Cao, Z., Simon, T., Wei, S.-E. & Sheikh, Y. Realtime multi-person 2D pose estimation using part affinity fields. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 1302–1310 (2017).
33. Simon, T., Joo, H., Matthews, I. & Sheikh, Y. Hand keypoint detection in single images using multiview bootstrapping. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 4645–4653 (2017).
34. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 770–778 (2016).
35. Dong, H. et al. TensorLayer: a versatile library for efficient deep learning development. *Proceedings of the 25th ACM International Conference on Multimedia* 1201–1204 (2017).
36. Pereira, T. D. et al. Fast animal pose estimation using deep neural networks. *Nat. Methods* **16**, 117–125 (2019).
37. OpenCV. *Open Source Computer Vision Library*, https://opencv.org (2015).
38. Lucas, B. D. & Kanade, T. An iterative image registration technique with an application to stereo vision. *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vol. 2, 674–679 (Morgan Kaufmann, 1981).
39. Oliphant, T. E. Python for scientific computing. *Comput. Sci. Eng.* **9**, 10–20 (2007).
40. Abadi, M. et al. TensorFlow: a system for large-scale machine learning. Preprint at https://arxiv.org/abs/1605.08695 (2016).
41. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, 2 (2014).
42. Mathis, M. W., Mathis, A. & Uchida, N. Somatosensory cortex plays an essential role in forelimb motor adaptation in mice. *Neuron* **93**, 1493–1503.e6 (2017).
43. McKinney, W. Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference* 51–56 (2010).
44. Hunter, J. D. Matplotlib: a 2D graphics environment. *Comput. Sci. Eng.* **9**, 90–95 (2007).
45. Durbin, J. & Koopman, S. J. *Time Series Analysis by State Space Methods* Vol. 38 (Oxford University Press, 2012).
46. Seabold, S. & Perktold, J. Statsmodels: econometric and statistical modeling with Python. *Proceedings of the 9th Python in Science Conference* 57–61 (2010).
47. Kabra, M., Robie, A. A., Rivera-Alba, M., Branson, S. & Branson, K. JAABA: interactive machine learning for automatic annotation of animal behavior. *Nat. Methods* **10**, 64 (2012).
48. Berman, G. J., Choi, D. M., Bialek, W. & Shaevitz, J. W. Mapping the stereotyped behaviour of freely moving fruit flies. *J. R. Soc. Interface* **11**, (2014).

49. Fox, E., Jordan, M. I., Sudderth, E. B. & Willsky, A. S. Sharing features among dynamical systems with beta processes. In *Advances in Neural Information Processing Systems* (eds Bengio, Y. et al.) **22**, 549–557 (Neural Information Processing Systems Foundation, 2009).
50. Wiltschko, A. B. et al. Mapping sub-second structure in mouse behavior. *Neuron* **88**, 1121–1135 (2015).
51. Vogelstein, J. T. et al. Discovery of brainwide neural-behavioral maps via multiscale unsupervised structure learning. *Science* **344**, 386–392 (2014).
52. Priebe, C. E., Marchette, D. J. & Healy, D. M. Integrated sensing and processing for statistical pattern recognition. *Mod. Signal Process.* **46**, 223 (2003).
53. Todd, J. G., Kain, J. S. & de Bivort, B. L. Systematic exploration of unsupervised methods for mapping behavior. *Phys. Biol.* **14**, 015002 (2017).
54. Russakovsky, O. et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* **115**, 211–252 (2015).
55. Pouw, W., de Jonge-Hoekstra, L. & Dixon, J. A. Stabilizing speech production through gesture-speech coordination. Preprint at https://psyarxiv.com/arzne (2018).
56. Wickens, A. et al. Magnetoelectric materials for miniature, wireless neural stimulation at therapeutic frequencies. Preprint at https://www.biorxiv.org/content/10.1101/461855v1 (2018).
57. Wilson, A. M. et al. Locomotion dynamics of hunting in wild cheetahs. *Nature* **498**, 185–189 (2013).
58. Jackson, B. E., Evangelista, D. J., Ray, D. D. & Hedrick, T. L. 3D for the people: multi-camera motion capture in the field with consumer-grade cameras and open source software. *Biol. Open* **5**, 1334–1342 (2016).
59. Urban, S., Leitloff, J. & Hinz, S. Improved wide-angle, fisheye and omnidirectional camera calibration. *ISPRS J. Photogramm. Remote Sens* **108**, 72–79 (2015).
60. Theriault, D. H. et al. A protocol and calibration method for accurate multi-camera field videography. *J. Exp. Biol.* **217**, 1843–1848 (2014).

Acknowledgements

DeepLabCut is an open-source tool on GitHub and has benefited from suggestions and edits by many individuals, including R. Eichler, J. Rauber, R. Warren, T. Abe, H. Wu, and J. Saunders. In particular, the authors thank R. Eichler for input on the modularized version. The authors thank the members of the Bethge Lab for providing the initial version of the Docker container. We also thank M. Li, J. Li, and D. Robson for use of the zebrafish image; B. Rogers for use of the horse images; and K. Cury for the fly images. The authors are grateful to E. Insafutdinov and C. Lassner for suggestions on how to best use the TensorFlow implementation of DeeperCut. We also thank A. Hoffmann, P. Mamidanna, and G. Kane for comments throughout this project. Last, the authors thank the Ann van Dyk Cheetah Centre (Pretoria, South Africa) for kindly providing access to their cheetahs. The authors thank NVIDIA Corporation for GPU grants to both M.W.M. and A.M. A.M. acknowledges a Marie Sklodowska-Curie International Fellowship within the 7th European Community Framework Program under grant agreement no. 622943. A.P. acknowledges an Oppenheimer Memorial Trust Fellowship and the National Research Foundation of South Africa (grant 99380). M.B. acknowledges funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) via the Collaborative Research Center (Projektnummer 276693517–SFB 1233: Robust Vision) and by the German Federal Ministry of Education and Research through the Tübingen AI Center (FKZ 01IS18039A). M.W.M. acknowledges a Rowland Fellowship from the Rowland Institute at Harvard.

Author contributions

Conceptualization: A.M., T.N., and M.W.M. A.M., T.N., and M.W.M. wrote the code. A.P. provided the cheetah data; A.C.C. labeled the cheetah data; A.C.C., A.M., and A.P. analyzed the cheetah data. M.W.M., A.M., and T.N. wrote the manuscript with input from all authors. M.W.M. and M.B. supervised the project.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41596-019-0176-0.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to MW.M.

Peer review information: *Nature Protocols* thanks Gonzalo G. de Polavieja and other anonymous reviewer(s) for their contribution to the peer review of this work.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 5 October 2018; Accepted: 9 April 2019;
Published online: 21 June 2019

Related link
Key reference using this protocol
Mathis, A. et al. *Nat. Neurosci.* **21**, 1281–1289 (2018): https://www.nature.com/articles/s41593-018-0209-y

nature research

Corresponding author(s): Mackenzie Mathis

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see Authors & Referees and the Editorial Policy Checklist.

Statistical parameters

When statistical analyses are reported, confirm that the following items are present in the relevant location (e.g. figure legend, table legend, main text, or Methods section).

n/a	Confirmed
☒	☐ The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☒	☐ An indication of whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☒	☐ The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☒	☐ A description of all covariates tested
☒	☐ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☒	☐ A full description of the statistics including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐ For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☒	☐ Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated
☒	☐ Clearly defined error bars *State explicitly what error bars represent (e.g. SD, SE, CI)*

Our web collection on statistics for biologists may be useful.

Software and code

Policy information about availability of computer code

Data collection	Data collected was previously described in Mathis et al, 2018 Nature Neuroscience. except for the new Cheetah example data where it is stated that it was taken with GoPro cameras, and proceeded with the DeepLabCut package.
Data analysis	All analysis and model generation code are deposited at https://github.com/AlexEMG/deeplabcut

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors/reviewers upon request. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

There is a Data Availability section that provides the link to the github.com code repository.

Field-specific reporting

Please select the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/authors/policies/ReportingSummary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	No predetermined sample size criteria was used.
Data exclusions	No data was excluded.
Replication	The original manuscript Mathis et al 2018 Nature Neuroscience describes the replication, namely model analysis was always performed with multiple randomized train/test splits, and run at least three times (i.e. see Figure 2) to validate the conclusions. All attempts at replication were successful.
Randomization	Data was randomly assigned to the training or test splits.
Blinding	Labelers were not told which videos or labeled data would be included in the training or test splits.

Reporting for specific materials, systems and methods

Materials & experimental systems

n/a	Involved in the study
☒	☐ Unique biological materials
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology
☐	☒ Animals and other organisms
☒	☐ Human research participants

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging

Animals and other organisms

Policy information about studies involving animals; ARRIVE guidelines recommended for reporting animal research

Laboratory animals	Mice: with a C57BL\6J background genotype of both sexes (female and male) were used, aged P60 to P360. All mice experiments were conducted with IUCAC approval at Harvard. For the Drosophila melanogaster, experiments were performed on adult females that were 5-6 days post-eclosion (Canton-S strain).
Wild animals	Cheetahs (n=3) were filmed at the Ann van Dyk Cheetah Centre (Prestoria, South Africa). They were filmed during their exercises runs. No cheetahs were sacrificed for this study.
Field-collected samples	The study did not invlove field-collected samples.