DeepLabCut User Guide (for single animal projects)

Contents

This document covers single/standard DeepLabCut use. If you have a complicated multi-animal scenario (i.e., they look the same), then please see our maDLC user guide.

To get started, you can use the GUI, or the terminal. See below.

DeepLabCut Project Manager GUI (recommended for beginners)

GUI: Simply `python -m deeplabcut` or MacOS: `pythonw -m deeplabcut`. The below functions are available to you in an easy-to-use graphical user interface. While most functionality is available, advanced users might want the additional flexibility that command line interface offers. Read more here.

As a reminder, the core functions are described in our Nature Protocols paper (published at the time of 2.0.6). Additional functions and features are continually added to the package. Thus, we recommend you read over the protocol and then please look at the following documentation and the doctrings. Thanks for using DeepLabCut!

DeepLabCut in the Terminal:

To begin, navigate to anaconda prompt and right-click to “open as admin “(windows), or simply launch “terminal” (unix/MacOS) on your computer. We assume you have DeepLabCut installed (if not, go here). Next, launch your conda env (i.e., for example `conda activate DLC-CPU`) and then type (windows/unix) `ipython` or (macOS) `pythonw`. Then type `import deeplabcut`.

(A) Create a New Project

The function **create_new_project** creates a new project directory, required subdirectories, and a basic project configuration file. Each project is identified by the name of the project (e.g. Reaching), name of the experimenter (e.g. YourName), as well as the date at 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.

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 entire video to the video directory (if the videos remain at the original location).

```
deeplabcut.create_new_project('Name of the project', 'Name of the experimenter',
['Full path of video 1', 'Full path of video2', 'Full path of video3'],
working_directory='Full path of the working directory', copy_videos=True/False,
multianimal=True/False)
```

Important path formatting note

Windows users, you must input paths as: `r'C:\Users\computername\Videos\reachingvideo1.avi'` or

`'C:\\Users\\computername\\Videos\\reachingvideo1.avi'`

(TIP: you can also place `config_path` in front of `deeplabcut.create_new_project` to create a variable that holds the path to the config.yaml file, i.e. `config_path=deeplabcut.create_new_project(...)`)

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. The project directory will have subdirectories: **dlc-models**, **labeled-data**, **training-datasets**, and **videos**. All the outputs generated during the course of a project will be stored in one of these subdirectories, thus allowing each project to be curated in separation from other projects. The purpose of the subdirectories is as follows:

dlc-models: This directory contains the subdirectories *test* and *train*, each of which holds the meta information with regard to the parameters of the feature detectors in configuration files. 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 case the training was interrupted.

labeled-data: This directory will store the frames used to create the training dataset. 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 every 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: 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`. Additionally, 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.

```
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)
```

*Please note, *Full path of the project configuration file* will be referenced as `config_path` throughout this protocol.

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 including their description can be found in Box1.

The `create_new_project` step writes the following parameters to the configuration file: *Task*, *scorer*, *date*, *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 adding new videos or manually removing videos.

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 step.

Parameters set during the project creation:

task: Name of the project (e.g. mouse-reaching). (do not edit)
scorer: Name of the experimenter (do not edit)
date: Date of creation of the project. (do not edit)
project_path: Full path of the project; 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.
(use the function add_new_videos to add more videos to the project; if necessary the paths can be edited manually, and the crop values are designed to 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 hand, Finger1, Finger2, Joystick. 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: It 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 iterations used to create the training dataset. The first iteration starts with 0 and thus the default value is set to 0.
Do not change this manually.

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] is the full video. The default is set to 0.
stop: Same as start, but 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] to split the dataset into training and testing dataset. The default is set to 0.95.
resnet: This specifies which pre-trained model to use. The default is set to 50 (user can choose 50 or 101, see also Mathis et al, 2018).

Used during video analysis:

batch_size: This specifies how many frames to process at once during inference (For tuning of this parameter see Mathis & Warren 2018).
snapshotindex: This specifies which checkpoint to use to evaluate the network. The default is set to -1. Use *"all"* to evaluate all the checkpoints.
Snapshots refer to the stored TensorFlow configuration, which holds the weights of the feature detectors.
p-cutoff: This specifies the threshold of the likelihood and helps distinguishing likely body parts from uncertain ones. The default is set to 0.1.
cropping: Specifies if the analysis video needs to be cropped. The default is set to 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 (due to the location refinement shifts).
This binary parameter makes sure 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 set to True.

corner2move2: This is the target location, if move2corner is True. The default is set to (50,50).

API Docs

> **Click the button to see API Docs** Click to show

(B) Configure the Project

Next, open the **config.yaml** file, which was created during **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, in particular you **must add the list of *bodyparts* (or points of interest)** that you want to track. You can also set the *colormap* here that is used for all downstream steps (can also be edited at anytime), like labeling GUIs, videos, etc. Here any matplotlib colormaps will do! Please DO NOT have spaces in the names of bodyparts.

bodyparts: are the bodyparts of each individual (in the above list).

(C) Data Selection (extract frames)

CRITICAL: A good training dataset should consist of a sufficient number of frames that capture the breadth of the behavior. This ideally implies to select the frames from different (behavioral) sessions, different lighting and different animals, if those vary substantially (to train an invariant, robust feature detector). Thus for creating a robust network that you can reuse in the laboratory, a good training dataset should reflect the diversity of the behavior with respect to postures, luminance conditions, background conditions, animal identities,etc. of the data that will be analyzed. For the simple lab behaviors comprising mouse reaching, open-field behavior and fly behavior, 100−200 frames gave good results Mathis et al, 2018. However, depending on the required accuracy, the nature of behavior, the video quality (e.g. motion blur, bad lighting) and the context, more or less frames might be necessary to create a good network. Ultimately, in

order to scale up the analysis to large collections of videos with perhaps unexpected conditions, one can also refine the data set in an adaptive way (see refinement below).

The function `extract_frames` extracts frames from all the videos in the project configuration file in order to create a training dataset. The extracted frames from all the videos are stored in a separate subdirectory named after the video file's name under the 'labeled-data'. This function also has various parameters that might be useful based on the user's need.

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

CRITICAL POINT: It is advisable to keep the frame size small, as large frames increase the training and inference time. The cropping parameters for each video can be provided in the config.yaml file (and see below). When running the function extract_frames, if the parameter crop=True, then you will be asked to draw a box within the GUI (and this is written to the config.yaml file).

`userfeedback` allows the user to check which videos they wish to extract frames from. In this way, if you added more videos to the config.yaml file it does not, by default, extract frames (again) from every video. If you wish to disable this question, set `userfeedback = True`.

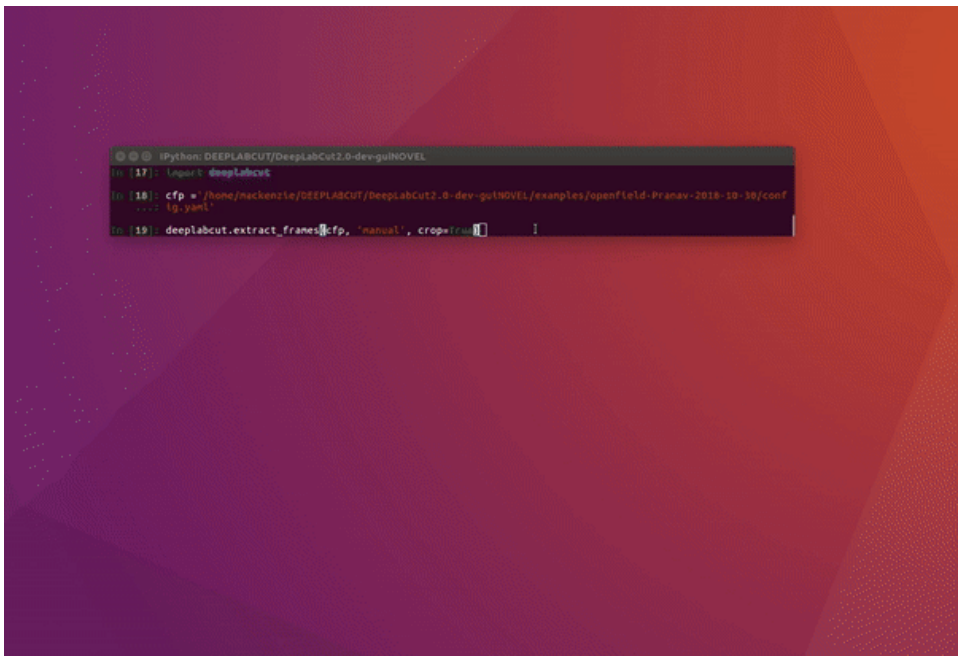
The provided function either selects frames from the videos that are randomly sampled from a uniform distribution (uniform), by clustering based on visual appearance (k-means), or by manual selection. Random selection of frames works best for behaviors where the postures vary across the whole video. However, some behaviors might be sparse, as in the case of reaching where the reach and pull are very fast and the mouse is not moving much between trials (thus, we have the default set to True, as this is best for most use-cases we encounter). In such a case, the function that allows selecting frames based on k-means derived quantization would be useful. If the user chooses to use k-means as a method to cluster the frames, then this function downsamples the video and clusters the frames using k-means, where each frame is treated as a vector. Frames from different clusters are then selected. This procedure makes sure that the frames look different. However, on large and long videos, this code is slow due to computational complexity.

CRITICAL POINT: It is advisable to extract frames from a period of the video that contains interesting behaviors, and not extract the frames across the whole video. 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 using the numframes2extract in the config.yaml file.

However, picking frames 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 the user feels specific frames are lacking, they can extract hand selected frames of interest using the interactive GUI provided along with the toolbox. This can be launched by using:

```
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 *Grab a Frame* (or a range of frames, as of version 2.0.5) to extract the frame(s). The user can also look at the extracted frames and e.g. delete frames (from the directory) that are too similar before reloading the set and then manually annotating them.



API Docs

> **Click the button to see API Docs** Click to show >

(D) Label Frames

The toolbox provides a function **label_frames** which helps the user to easily label all the extracted frames using an interactive graphical user interface (GUI). The user should have already named the body parts to label (points of interest) in the project's configuration file by providing a list. The following command invokes the labeling toolbox.

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

The user needs to use the *Load Frames* button to select the directory which stores the extracted frames from one of the videos. Subsequently, the user can use one of the radio buttons (top right) to select a body part to label. RIGHT click to add the label. Left click to drag the label, if needed. If you label a part accidentally, you can use the middle button on your mouse to delete! If you cannot see a body part in the frame, skip over the label! Please see the `HELP` button for more user instructions. This auto-advances once you labeled the first body part. You can also advance to the next frame by clicking on the RIGHT arrow on your keyboard (and go to a previous frame with LEFT arrow). Each label will be plotted as a dot in a unique color.

The user is free to move around the body part and once satisfied with its position, can select another radio button (in the top right) to switch to the respective body part (it otherwise auto-advances). The user can skip a body part if it is not visible. Once all the visible body parts are labeled, then the user can use 'Next Frame' to load the following frame. The user needs to save the labels after all the frames from one of the videos are labeled by clicking the save button at the bottom right. Saving the labels will create a labeled dataset for each video in a hierarchical data file format (HDF) in the subdirectory corresponding to the particular video in **labeled-data**. You can save at any intermediate step (even without closing the GUI, just hit save) and you return to labeling a dataset by reloading it!

CRITICAL POINT: 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 by the user. They can simply be skipped by not applying the label anywhere on the frame.

OPTIONAL: In the event of adding more labels to the existing labeled dataset, the user need to append the new labels to the bodyparts in the config.yaml file. Thereafter, the user can call the function **label_frames**. As of 2.0.5+: then a box will pop up and ask the user if they wish to display all parts, or only add in the new labels. Saving the labels after all the images are labelled will append the new labels to the existing labeled dataset.

HOT KEYS IN THE Labeling GUI (also see "help" in GUI):

```
Ctrl + C: Copy labels from previous frame. With multi-animal DLC, only the keypoints
of the animal currently selected are duplicated.
Keyboard arrows: advance frames
delete key: delete label
```

API Docs

> **Click the button to see API Docs** Click to show

(E) Check Annotated Frames

OPTIONAL: Checking if the labels were created and stored correctly is beneficial for training, since labeling is one of the most critical parts for creating the training dataset. The DeepLabCut toolbox provides a function 'check_labels' to do so. It is used as follows:

```
deeplabcut.check_labels(config_path, visualizeindividuals=True/False)
```

For each video directory in labeled-data this function creates a subdirectory with **labeled** as a suffix. Those directories contain the frames plotted with the annotated body parts. The user can double check if the body parts are labeled correctly. If they are not correct, the user can reload the frames (i.e. `deeplabcut.label_frames`), move them around, and click save again.

API Docs

> **Click the button to see API Docs** Click to show

(F) Create Training Dataset(s)

CRITICAL POINT: Only run this step **where** you are going to train the network. If you label on your laptop but move your project folder to Google Colab or AWS, lab server, etc, then run the step below on that platform! If you labeled on a Windows machine but train on Linux, this is fine as of 2.0.4 onwards it will be done automatically (it saves file sets as both Linux and Windows for you).

- If you move your project folder, you must **only** change the `project_path` in the main config.yaml file - that's it - no need to change the video paths, etc! Your project is fully portable.
- If you run this on the cloud, before importing `deeplabcut` you need to suppress GUIs. As you can see in our [demo notebooks]((https://github.com/DeepLabCut/DeepLabCut/blob/master/examples/COLAB_DEMO_mouse_openfield.ipynb) for running DLC training, evaluation, and novel video analysis on the Cloud, you must first suppress GUIs - server computers don't have a screen you can interact with. So, before you launch ipython, run `export DLClight=True` (see more tips in the full PDF user-guide).

OVERVIEW: This function combines the labeled datasets from all the videos and splits them to create train and test datasets. The training data will be used to train the network, while the test data set will be used for evaluating the network. The function **create_training_dataset** performs those steps.

```
deeplabcut.create_training_dataset(config_path, augmenter_type='imgaug')
```

- OPTIONAL: If the user wishes to benchmark the performance of the DeepLabCut, they can create multiple training datasets by specifying an integer value to the `num_shuffles`; see the docstring for more details.
- Each iteration of the creation of a training dataset will create a `.mat` file, which is used by the feature detectors, and a `.pickle` file that contains the meta information about the training dataset. This also creates two subdirectories within **dlc-models** called `test` and `train`, and these each have a configuration file called pose_cfg.yaml. Specifically, the user

can edit the **pose_cfg.yaml** within the **train** subdirectory before starting the training. These configuration files contain meta information with regard to the parameters of the feature detectors. Key parameters are listed in Box 2.

- At this step, the ImageNet pre-trained networks (i.e. ResNet-50, ResNet-101 and ResNet-152, etc) weights will be downloaded. If they do not download (you will see this downloading in the terminal, then you may not have permission to do so (something we have seen with some Windows users - see the **docs for more help!**).

CRITICAL POINT: At this step, for **create_training_dataset** you select the network you want to use, and any additional data augmentation (beyond our defaults). You can set `net_type` and `augmenter_type` when you call the function.

DATA AUGMENTATION: At this stage you can also decide what type of augmentation to use. The default loaders work well for most all tasks (as shown on www.deeplabcut.org), but there are many options, more data augmentation, intermediate supervision, etc. Please look at the **pose_cfg.yaml** file for a full list of parameters **you might want to change before running this step.** There are several data loaders that can be used. For example, you can use the default loader (introduced and described in the Nature Protocols paper), TensorPack for data augmentation (currently this is easiest on Linux only), or imgaug. We recommend `imgaug`. You can set this by passing:`deeplabcut.create_training_dataset(config_path, augmenter_type='imgaug')`

The differences of the loaders are as follows:

- `imgaug`: a lot of augmentation possibilities, efficient code for target map creation & batch sizes >1 supported. You can set the parameters such as the `batch_size` in the `pose_cfg.yaml` file for the model you are training. This is the recommended DEFAULT!
- `crop_scale`: our standard DLC 2.0 introduced in Nature Protocols variant (scaling, auto-crop augmentation)
- `tensorpack`: a lot of augmentation possibilities, multi CPU support for fast processing, target maps are created less efficiently than in imgaug, does not allow batch size>1
- `deterministic`: only useful for testing, freezes numpy seed; otherwise like default.

Alternatively, you can set the loader (as well as other training parameters) in the **pose_cfg.yaml** file of the model that you want to train. Note, to get details on the options, look at the default file: **pose_cfg.yaml**.

MODEL COMPARISON: You can also test several models by creating the same test/train split for different networks. You can easily do this in the Project Manager GUI, or use the function `deeplabcut.create_training_model_comparison`.

Please also consult the following page on selecting models: https://deeplabcut.github.io/DeepLabCut/docs/recipes/nn.html#what-neural-network-should-i-use-trade-offs-speed-performance-and-considerations

See Box 2 on how to specify **which network is loaded for training (including your own network, etc):**

BOX 2: Parameters of interest in the network configuration file, pose_cfg.yaml

net_type: Here you can set what neural network you want to deploy. The default is resnet_50. Current options include resnet_50, resnet_101, resnet152, mobilenet_v2_1.0, mobilenet_v2_0.75 mobilenet_v2_0.5, mobilenet_v2_0.35. If you pick resnet_101 or 152, please set **intermediate_supervision** to True for best performance.

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 >90MB, 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 (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 user wants to stop before 1M, 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 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 & Warren, 2018.

The following parameters allow one to change the network resolution:

global_scale: all images in the dataset will be re-scaled by the following scaling factor to be processed by the CNN. You can select the optimal scale by cross-validation (See discussion in Mathis et al, 2018). Default is 0.8.

pos_dist_thresh: all locations within this distance threshold in pixels are considered positive training samples for detector (See discussion in Mathis et al 2018). Default is 17. For very small objects, such as a fly leg in a large image, you should consider lowering this value.

The following parameters modulate the data augmentation:

dataset_type: this is used for additional augmentation. Currently, this supports default augmentation, tensorpack, imgaug, and deterministic data loading/augmentation. The default is our standard pose loader (called default). *If you want to pass this when yu run `create_training_dataset`, you can pass `augmenter_type='default'`, etc).

During training each image will be randomly re-scaled 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 training dataset is symmetric around vertical axis, this Boolean variable allows random augmentation. Default is false.

cropping: Automatically crop images during training. Default is True.

cropratio: Fraction of training samples that are cropped. Default is 40%

minsize, leftwidth, rightwidth, bottomheight, topheight: Define dimensions and limits for cropping.

API Docs for deeplabcut.create_training_dataset

Click the button to see API Docs Click to show

API Docs for deeplabcut.create_training_model_comparison

Click the button to see API Docs Click to show

(G) Train The Network

The function 'train_network' helps the user in training the network. It is used as follows:

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

The set of arguments in the function starts training the network for the dataset created for one specific shuffle. Note that you can change the loader (imgaug/default/etc) as well as other training parameters in the **pose_cfg.yaml** file of the model that you want to train (before you start training).

Example parameters that one can call:

```
deeplabcut.train_network(config_path, shuffle=1, trainingsetindex=0, gputouse=None,
max_snapshots_to_keep=5, autotune=False, displayiters=100, saveiters=15000,
maxiters=30000, allow_growth=True)
```

By default, the pretrained networks are not in the DeepLabCut toolbox (as they are around 100MB each), but they get downloaded before you train. However, if not previously downloaded from the TensorFlow model weights, it will be downloaded and stored in a subdirectory *pre-trained* under the subdirectory *models* in *Pose_Estimation_Tensorflow*. At user specified iterations during training checkpoints are stored in the subdirectory *train* under the respective iteration directory.

If the user wishes to restart the training at a specific checkpoint they can specify the full path of the checkpoint to the variable `init_weights` in the **pose_cfg.yaml** file under the *train* subdirectory (see Box 2).

CRITICAL POINT: It is recommended to train the ResNets or MobileNets for thousands of iterations until the loss plateaus (typically around **500,000**) if you use batch size 1. If you want to batch train, we recommend using Adam, see more here: https://deeplabcut.github.io/DeepLabCut/docs/recipes/nn.html#using-custom-image-augmentation.

The variables `display_iters` and `save_iters` in the **pose_cfg.yaml** file allows the user to alter how often the loss is displayed and how often the weights are stored.

maDeepLabCut CRITICAL POINT: For multi-animal projects we are using not only different and new output layers, but also new data augmentation, optimization, learning rates, and batch training defaults. Thus, please use a lower `save_iters` and `maxiters`. I.e. we suggest saving every 10K-15K iterations, and only training until 50K-100K iterations. We recommend you look closely at the loss to not overfit on your data. The bonus, training time is much less!!!

API Docs

> **Click the button to see API Docs** Click to show

(H) Evaluate the Trained Network

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 due to the probabilistic output of the scoremap, it can, if sufficiently trained, also reliably report if a body part is visible in a given frame. (see discussions of finger tips in reaching and the Drosophila legs during 3D behavior in [Mathis et al, 2018]). The evaluation results are computed by typing:

```
deeplabcut.evaluate_network(config_path,Shuffles=[1], plotting=True)
```

Setting `plotting` to true plots all the testing and training frames with the manual and predicted labels. The user should visually check the labeled test (and training) images that are created in the 'evaluation-results' directory. Ideally, DeepLabCut labeled unseen (test images) according to the user's required accuracy, and the average train and test errors are comparable (good generalization). What (numerically) comprises an acceptable MAE depends on many factors (including the size of the tracked body parts, the labeling variability, etc.). Note that the test error can also be larger than the training error due to human variability (in labeling, see Figure 2 in Mathis et al, Nature Neuroscience 2018).

Optional parameters:

```
  Shuffles: list, optional -List of integers specifying the shuffle indices of the
training dataset. The default is [1]

  plotting: bool, optional -Plots the predictions on the train and test images. The
default is `False`; if provided it must be either `True` or `False`

  show_errors: bool, optional -Display train and test errors. The default is `True`

  comparisonbodyparts: list of bodyparts, Default is all -The average error will be
computed for those body parts only (Has to be a subset of the body parts).

  gputouse: int, optional -Natural number indicating the number of your GPU (see
number in nvidia-smi). If you do not have a GPU, put None. See:
https://nvidia.custhelp.com/app/answers/detail/a_id/3751/~/useful-nvidia-smi-queries
```

The plots can be customized by editing the **config.yaml** file (i.e., the colormap, scale, 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 plus ('+'), DeepLabCut's predictions either as '.' (for confident predictions with likelihood > p-cutoff) and 'x' for (likelihood <= `pcutoff`).

The evaluation results for each shuffle of the training dataset are stored in a unique subdirectory in a newly created directory 'evaluation-results' in the project directory. The user can visually inspect if the distance between the labeled and the predicted body parts are acceptable. In the event of benchmarking with different shuffles of same training dataset, the user can provide multiple shuffle indices to evaluate the corresponding network. Note that with multi-animal projects additional distance statistics aggregated over animals or bodyparts are also stored in that directory. This aims at providing a finer quantitative evaluation of multi-animal prediction performance before animal tracking. If the generalization is not sufficient, the user might want to:

• check if the labels were imported correctly; i.e., invisible points are not labeled and the points of interest are labeled accurately

• make sure that the loss has already converged

• consider labeling additional images and make another iteration of the training data set

OPTIONAL: You can also plot the scoremaps, locref layers, and PAFs:

```
deeplabcut.extract_save_all_maps(config_path, shuffle=shuffle, Indices=[0, 5])
```

you can drop "Indices" to run this on all training/testing images (this is slow!)

API Docs

> **Click the button to see API Docs** Click to show

(I) Novel Video Analysis:

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 enter the corresponding index of the checkpoint to the variable snapshotindex in the config.yaml file. By default, the most recent checkpoint (i.e. last) is used for analyzing the video. Novel/new videos **DO NOT have to be in the config file!** You can analyze new videos anytime by simply using the following line of code:

```
deeplabcut.analyze_videos(config_path,
['fullpath/analysis/project/videos/reachingvideo1.avi'], save_as_csv=True)
```

There are several other optional inputs, such as:

```
deeplabcut.analyze_videos(config_path, videos, videotype='avi', shuffle=1,
trainingsetindex=0, gputouse=None, save_as_csv=False, destfolder=None, dynamic=(True,
.5, 10))
```

The labels are stored in a MultiIndex Pandas Array, 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 Hierarchical Data Format (HDF) 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 comma-separated values format (.csv), which in turn can be imported in many programs, such as MATLAB, R, Prism, etc.; This flag is set to `False` by default. You can also set a destination folder (`destfolder`) for the output files by passing a path of the folder you wish to write to.

API Docs

Click the button to see API Docs Click to show

Novel Video Analysis: extra features

Dynamic-cropping of videos:

As of 2.1+ we have a dynamic cropping option. Namely, if you have large frames and the animal/object occupies a smaller fraction, you can crop around your animal/object to make processing speeds faster. For example, if you have a large open field experiment but only track the mouse, this will speed up your analysis (also helpful for real-time applications). To use this simply add `dynamic=(True,.5,10)` when you call `analyze_videos`.

```
dynamic: triple containing (state, detectiontreshold, margin)

        If the state is true, then dynamic cropping will be performed. That means
that if an object is detected (i.e., any body part > detectiontreshold), then object
boundaries are computed according to the smallest/largest x position and
smallest/largest y position of all body parts. This window is expanded by the margin
and from then on only the posture within this crop is analyzed (until the object is
lost; i.e., <detectiontreshold). The current position is utilized for updating the
crop window for the next frame (this is why the margin is important and should be set
large enough given the movement of the animal).
```

(J) Filter pose data data (RECOMMENDED!):

You can also filter the predictions with a median filter (default) or with a SARIMAX model, if you wish. This creates a new .h5 file with the ending *_filtered* that you can use in create_labeled_data and/or plot trajectories.

```
deeplabcut.filterpredictions(config_path,
['fullpath/analysis/project/videos/reachingvideo1.avi'])
```

An example call:

```
deeplabcut.filterpredictions(config_path,['fullpath/analysis/project/videos'],
videotype='.mp4',filtertype= 'arima',ARdegree=5,MAdegree=2)
```

Here are parameters you can modify and pass:

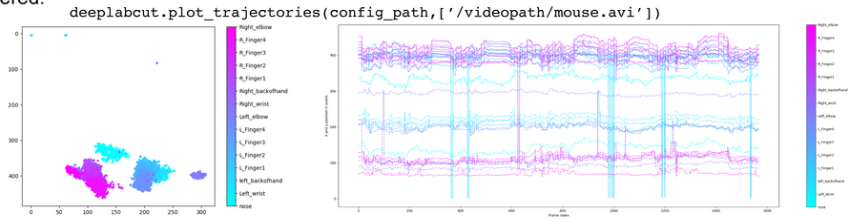
```
deeplabcut.filterpredictions(config_path,
['fullpath/analysis/project/videos/reachingvideo1.avi'], shuffle=1,
trainingsetindex=0, comparisonbodyparts='all', filtertype='arima', p_bound=0.01,
ARdegree=3, MAdegree=1, alpha=0.01)
```

Here is an example of how this can be applied to a video:

```
deeplabcut.filterpredictions(config_path,['/videopath/mouse.avi'], p_bound=0.1, ARdegree=5, MAdegree=1)
```

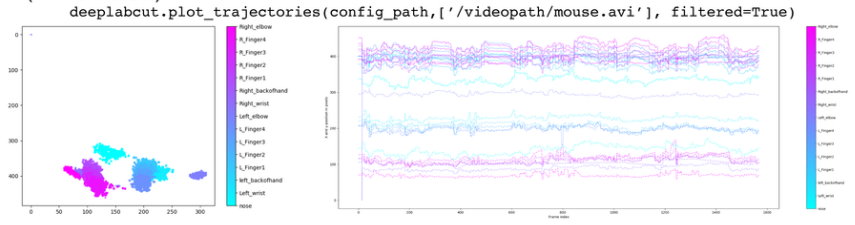
Unfiltered:

```
deeplabcut.plot_trajectories(config_path,['/videopath/mouse.avi'])
```



Filtered (ARIMA model):

```
deeplabcut.plot_trajectories(config_path,['/videopath/mouse.avi'], filtered=True)
```



API Docs

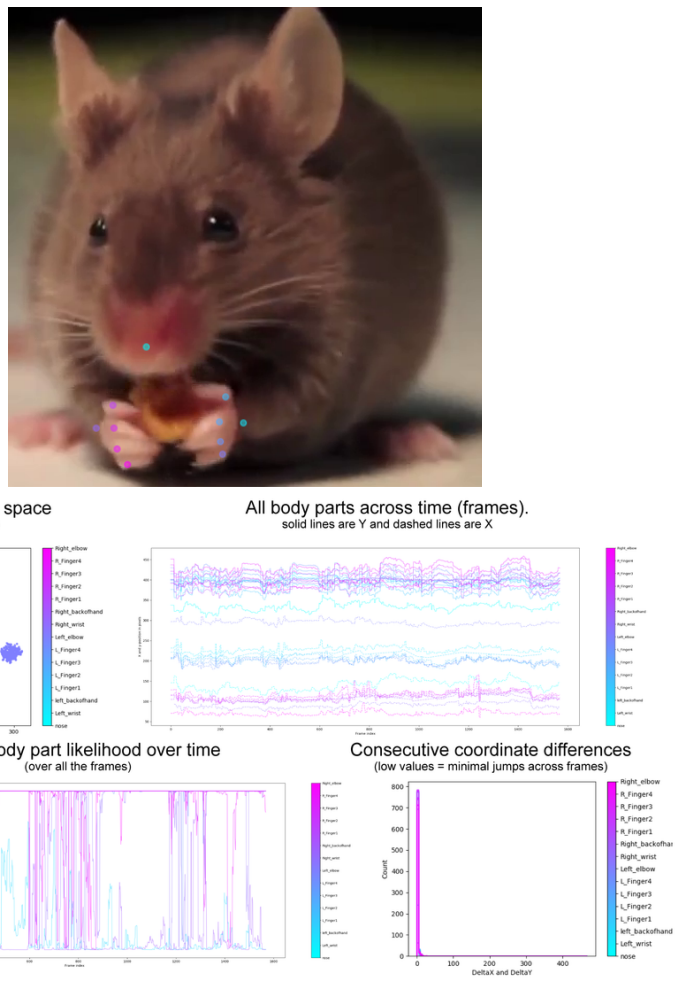
Click the button to see API Docs Click to show

(K) Plot Trajectories:

The plotting components of this toolbox utilizes matplotlib. Therefore, these plots can easily be customized by the end user. We also provide a function to plot the trajectory of the extracted poses across the analyzed video, which can be called by typing:

```
deeplabcut.plot_trajectories(config_path,
['fullpath/analysis/project/videos/reachingvideo1.avi'])
```

It creates a folder called `plot-poses` (in the directory of the video). The plots display the coordinates of body parts vs. time, likelihoods vs time, the x- vs. y- coordinate of the body parts, as well as 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). Here are example plot outputs on a demo video (left):



API Docs

Click the button to see API Docs Click to show

(L) Create Labeled Videos:

Additionally, 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. There are two modes to create videos: FAST and SLOW (but higher quality!). If you want to create high-quality videos, please add `save_frames=True`. One can use the command as follows to create multiple labeled videos:

```
deeplabcut.create_labeled_video(config_path,
['fullpath/analysis/project/videos/reachingvideo1.avi','fullpath/analysis/project/vid
eos/reachingvideo2.avi'], save_frames = True/False)
```

Optionally, if you want to use the filtered data for a video or directory of filtered videos pass `filtered=True`, i.e.:

```
deeplabcut.create_labeled_video(config_path, ['fullpath/afolderofvideos'],
videotype='.mp4', filtered=True)
```

You can also optionally add a skeleton to connect points and/or add a history of points for visualization. To set the “trailing points” you need to pass `trailpoints`:

```
deeplabcut.create_labeled_video(config_path, ['fullpath/afolderofvideos'],
videotype='.mp4', trailpoints=10)
```

To draw a skeleton, you need to first define the pairs of connected nodes (in the `config.yaml` file) and set the skeleton color (in the `config.yaml` file). There is also a GUI to help you do this, use by calling `deeplabcut.SkeletonBuilder(config+path)`!

Here is how the `config.yaml` additions/edits should look (for example, on the Openfield demo data we provide):

```
# Plotting configuration
skeleton: [['snout', 'leftear'], ['snout', 'rightear'], ['leftear', 'tailbase'],
['leftear', 'rightear'], ['rightear','tailbase']]
skeleton_color: white
pcutoff: 0.4
dotsize: 4
alphavalue: 0.5
colormap: jet
```

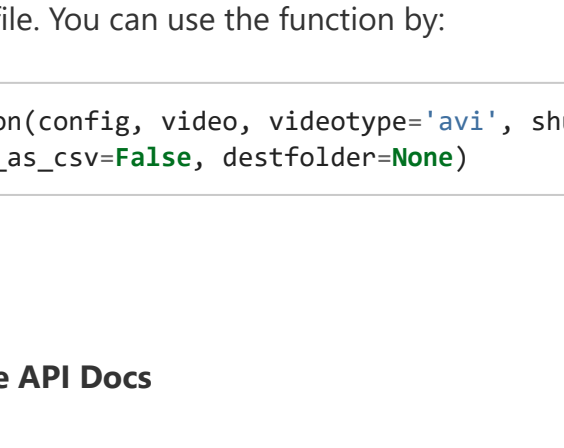
Then pass `draw_skeleton=True` with the command:

```
deeplabcut.create_labeled_video(config_path,['fullpath/afolderofvideos'],
videotype='.mp4', draw_skeleton = True)
```

NEW as of 2.2b8: You can create a video with only the "dots" plotted, i.e., in the [style of Johansson](), by passing `keypoints_only=True`:

```
deeplabcut.create_labeled_video(config_path,['fullpath/afolderofvideos'],
videotype='.mp4', keypoints_only=True)
```

PRO TIP: that the **best quality videos** are created when `save_frames=True` is passed. Therefore, when `trailpoints` and `draw_skeleton` are used, we **highly** recommend you also pass `save_frames=True`!



This function has various other parameters, in particular the user can set the `colormap`, the `dotsize`, and `alphavalue` of the labels in **config.yaml** file.

API Docs

> **Click the button to see API Docs** Click to show

Extract "Skeleton" Features:

NEW, as of 2.0.7+: You can save the "skeleton" that was applied in `create_labeled_videos` for more computations. Namely, it extracts length and orientation of each "bone" of the skeleton as defined in the **config.yaml** file. You can use the function by:

```
deeplabcut.analyzeskeleton(config, video, videotype='avi', shuffle=1,
trainingsetindex=0, save_as_csv=False, destfolder=None)
```

API Docs

> **Click the button to see API Docs** Click to show

(M) Optional Active Learning -> Network Refinement: Extract Outlier Frames

While DeepLabCut typically generalizes well across datasets, one might want to optimize its performance in various, perhaps unexpected, situations. For generalization to large data sets, images with insufficient labeling performance can be extracted, 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 2018). Then, due 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 could be eliminated by relabeling a few examples and then re-training. 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.

All this can be done for a specific video by typing (see other optional inputs below):

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

We provide various frame-selection methods for this purpose. In particular the user can set:

```
outlieralgorithm: 'fitting', 'jump', or 'uncertain'``
```

• select frames if the likelihood of a particular or all body parts lies below *pbound* (note this could also be due to occlusions rather than errors); (`outlieralgorithm='uncertain'`), but also set `p_bound`.

• select frames where a particular body part or all body parts jumped more than *uf* pixels from the last frame (`outlieralgorithm='jump'`).

• select frames if the predicted body part location deviates from a state-space model fit to the time series of individual body parts. Specifically, this method fits an Auto Regressive Integrated Moving Average (ARIMA) model to the time series for each body part. Thereby each body part detection with a likelihood smaller than pbound is treated as missing data. Putative outlier frames are then identified as time points, where the average body part estimates are at least *uf* pixel away from the fits. The parameters of this method are *uf*, *pbound*, the ARIMA parameters as well as the list of body parts to average over (can also be `all`).

• manually select outlier frames based on visual inspection from the user (`outlieralgorithm='manual'`).

As an example:

```
deeplabcut.extract_outlier_frames(config_path, ['videofile_path'],
outlieralgorithm='manual')
```

In general, depending on the parameters, these methods might return much more frames than the user wants to extract (`numframes2pick`). Thus, this list is then used to select outlier frames either by randomly sampling from this list (`extractionalgorithm='uniform'`), by performing `extractionalgorithm='k-means'` clustering on the corresponding frames.

In the automatic configuration, before the frame selection happens, the user is informed about the amount of frames satisfying the criteria and asked if the selection should proceed. This step allows the user to perhaps change the parameters of the frame-selection heuristics first (i.e. to make sure that not too many frames are qualified). The user can run the extract_outlier_frames iteratively, and (even) extract additional frames from the same video. Once enough outlier frames are extracted the refinement GUI can be used to adjust the labels based on user feedback (see below).

API Docs

Click the button to see API Docs Click to show

(N) Refine Labels: Augmentation of the Training Dataset

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

(A) Visible body part with accurate DeepLabCut prediction. These labels do not need any modifications.

(B) Visible body part but wrong DeepLabCut prediction. Move the label's location to the actual position of the body part.

(C) Invisible, occluded body part. Remove the predicted label by DeepLabCut with a middle 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).

(D) Invalid images: In the unlikely event that there are any invalid images, the user should remove such an image and their corresponding predictions, if any. Here, the GUI will prompt the user to remove an image identified as invalid.

The labels for extracted putative outlier frames can be refined by opening the GUI:

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

This will launch a GUI where the user can refine the labels.

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

After correcting the labels for all the frames in each of the subdirectories, the users should merge the data set to create a new dataset. In this step the iteration parameter in the config.yaml file is automatically updated.

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

Once the dataset is merged, the user can test if the merging process was successful by plotting all the labels (Step E). Next, with this expanded training set the user can now create a novel training set and train the network as described in Steps F and G. 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).

Now you can run `create_training_dataset`, then `train_network`, etc. If your original labels were adjusted at all, start from fresh weights (the typically recommended path anyhow), otherwise consider using your already trained network weights (see Box 2).

If after training the network generalizes well to the data, proceed to analyze new videos. Otherwise, consider labeling more data.

API Docs for deeplabcut.refine_labels

Click the button to see API Docs Click to show

API Docs for deeplabcut.merge_datasets

Click the button to see API Docs Click to show

Jupyter Notebooks for Demonstration of the DeepLabCut Workflow

We also provide two Jupyter notebooks for using DeepLabCut on both a pre-labeled dataset, and on the end user's own dataset. Firstly, we prepared an interactive Jupyter notebook called run_yourowndata.ipynb that can serve as a template for the user to develop a project. Furthermore, we provide a notebook for an already started project with labeled data. The example project, named as Reaching-Mackenzie-2018-08-30 consists of a project configuration file with default parameters and 20 images, which are cropped around the region of interest as an example dataset. These images are extracted from a video, which was recorded in a study of skilled motor control in mice. Some example labels for these images are also provided. See more details here.

3D Toolbox

Please see 3D overview for information on using the 3D toolbox of DeepLabCut (as of 2.0.7+).

Other functions, some are yet-to-be-documented:

We suggest you check out these additional helper functions, that could be useful (they are all optional).