DeepLabCut AI Residency

Day 2 Session 1:
Training & networks

July 30 & August 1, 2025

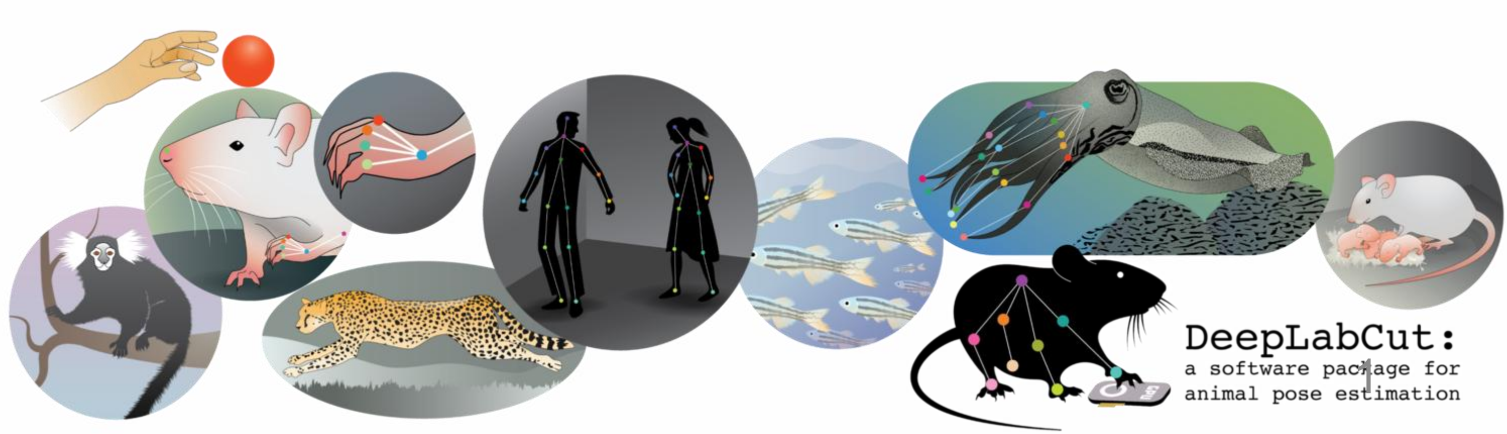
McGill University, Montreal

Jiayue Yang

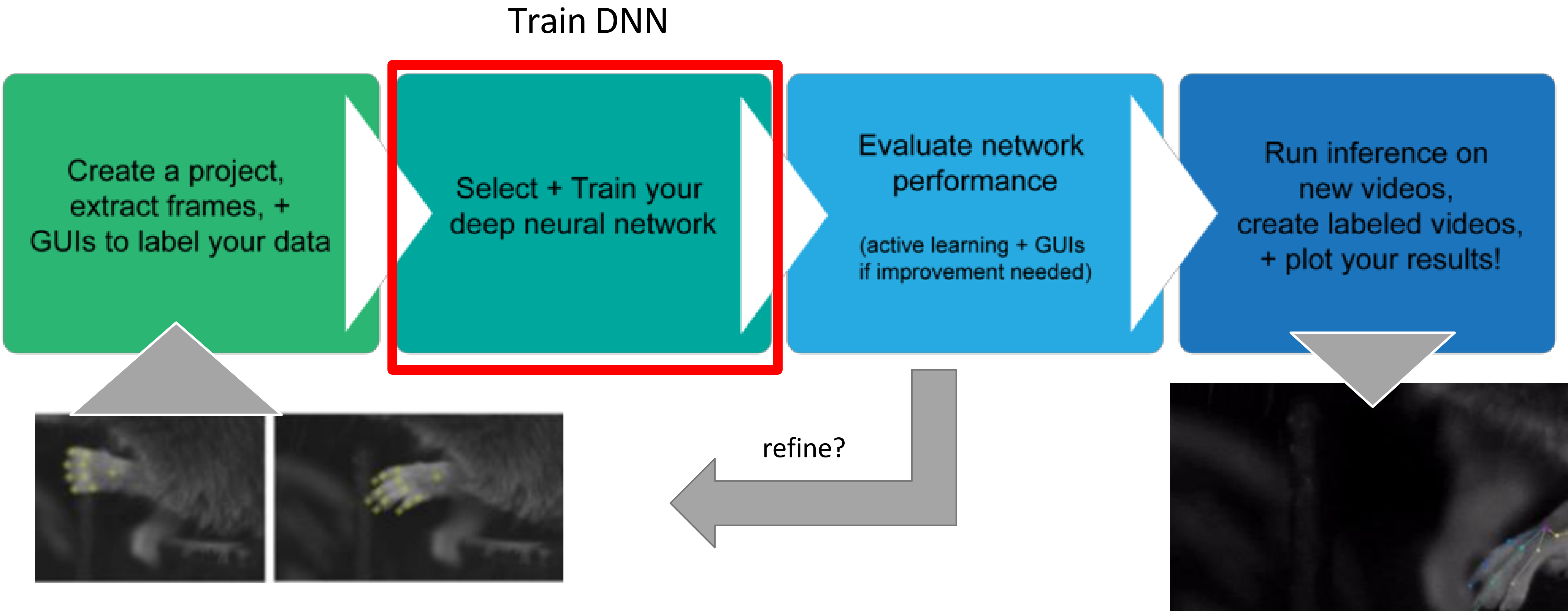
Vic Shao-Chinh Chiang

DeepLabCut AI Residency
next-gen animal behavior

McGill



Recall: DeepLabCut workflow



Nath et al. (2019). Nature Protocols.

Network architecture?

Network architecture:

- Network backbones: deep residual networks (**ResNet**)
- “Sees" and "understands" the image by identifying edges, shapes, and patterns
- Good at identifying complex features from raw images

Pre-training (Weights):

- ***ImageNet*** *pretrained (with basic “understanding”)*
- *Higher network number → slower but more powerful*
- DeepLabCut model zoo….

Multi-animal training:

- Adds extra steps to **separate different animals** (instance identification)
- More **complex** training and labeling are required

Augmentation & Training structures?

2.2 Data augmentation parameters

In the simplest form, we can think of data augmentation as something similar to imagination or dreaming. Humans imagine different scenarios based on experience, ultimately allowing us to better understand our world. 2, 3, 4

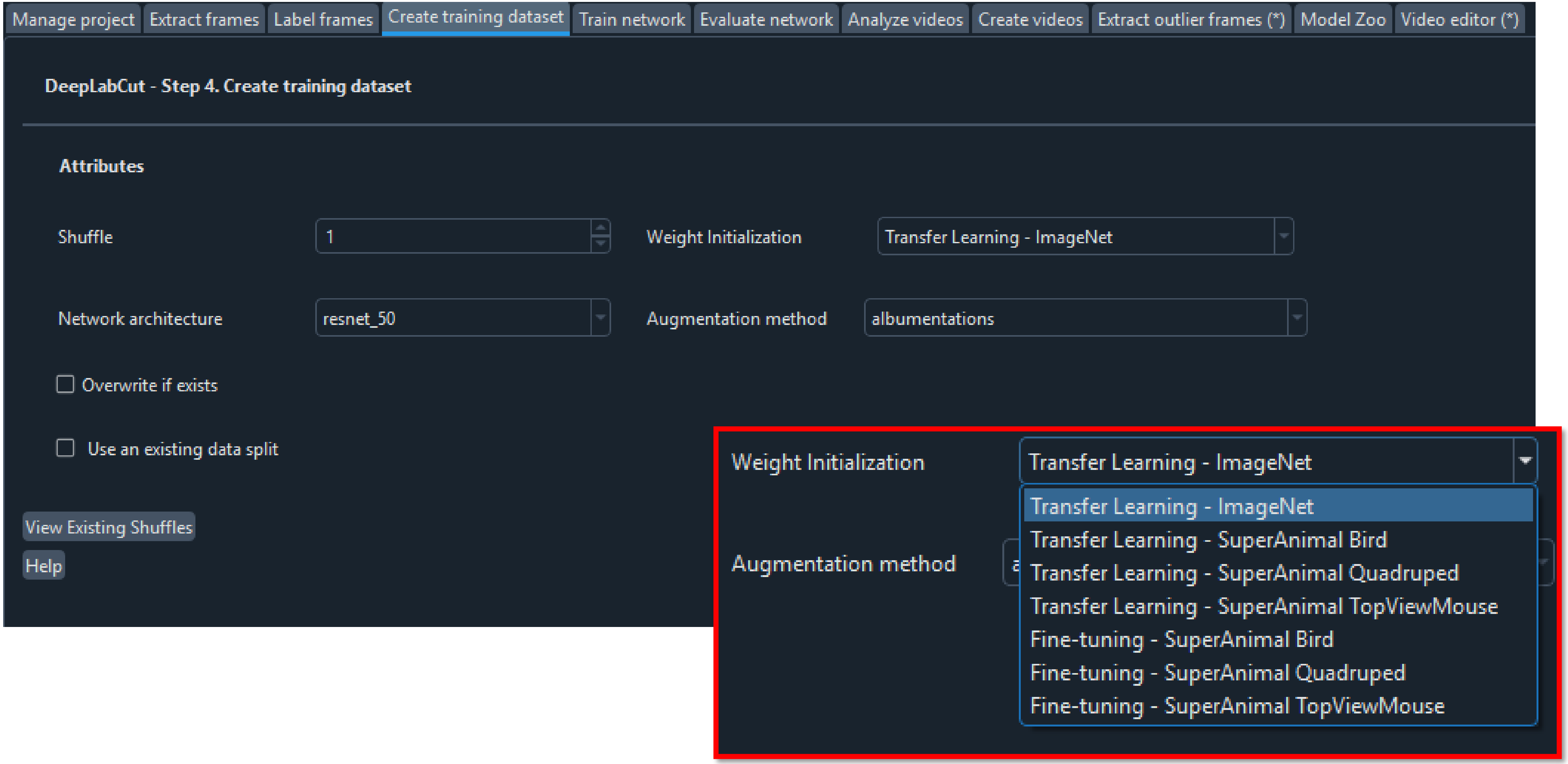
Similarly, we train our models to different types of "imagined" scenarios, which we limit to the foreseeable ones, so we ultimately get a robust model that can more likely handle new data and scenes.

Classes of data augmentations, characterized by their nature, are given by:

- **Geometric transformations**
    1. `scale_jitter_lo` and `scale_jitter_up` Different object sizes in scene
    2. `rotation` Handle head/body tilt, if your animal turns sideways
    3. `rotratio`
    4. `mirror` Symmetry recognition, left vs tight hand
    5. `crop_size`
    6. `crop_ratio` Remove unwanted parts of the images
    7. `max_shift` Max relative shift to the position of the crop centre
    8. `crop_sampling`
- **Kernel transformations** 9. `sharpening` and `sharpen_ratio` 10. `edge_enhancement`

Mathis et al. (2018). Nature Neuroscience.
Nath, Mathis et al. (2019). Nature Protocols.
Lauer et al. (2022). Nature Methods.
Ye et al. (2024). Nature Communications.

https://deeplabcut.github.io/DeepLabCut/docs/recipes/pose_cfg_file_breakdown.html#22-data-augmentation-parameters



Manage project
Extract frames
Label frames
Create training dataset
Train network
Evaluate network
Analyze videos
Create videos
Extract outlier frames (*)
Model Zoo
Video editor (*)
DeepLabCut - Step 4. Create training dataset
Attributes
Shuffle
1
Weight Initialization
Transfer Learning - ImageNet
Network architecture
resnet_50
Augmentation method
albumentations
Overwrite if exists
Use an existing data split
View Existing Shuffles
Help
Weight Initialization
Transfer Learning - ImageNet
Augmentation method
Transfer Learning - ImageNet
Transfer Learning - SuperAnimal Bird
Transfer Learning - SuperAnimal Quadruped
Transfer Learning - SuperAnimal TopViewMouse
Fine-tuning - SuperAnimal Bird
Fine-tuning - SuperAnimal Quadruped
Fine-tuning - SuperAnimal TopViewMouse

How to select the training network?

- **Performance vs. training speed**
  - Trade-off
- Backbone (**ResNet**): Extracts visual features from the image
  - Resnet 50 (balanced in speed and accuracy)
  - Resnet 101 (deeper, more powerful, but slower)
  - Mobilenet_v2_1.0 (lightweight, faster, less accurate)
- **Pose Estimation** Head
  - Converts features into scoremaps (heatmaps) for each body part
- **Cropping** and **downsample**
  - Remove or downsample irrelevant part of the image (e.g. no animal there, etc.)

Training hyperparameters

Hyperparameters	Typical values	Purposes	Why?
batch_size	1 - 8	# images processed before the model updates	Small sizes help with limited GPU memory; 1 is safest, 8 good for generalization
max_iters	200,000–500,000	How long and how many steps to train the model	More iterations = better learning, up to a point (i.e. plateau)
save_iters	10,000	How often to save a model checkpoint	Useful to create better model
display_iters	100, 500, etc.	How often to print training progress	Help checking training process

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.

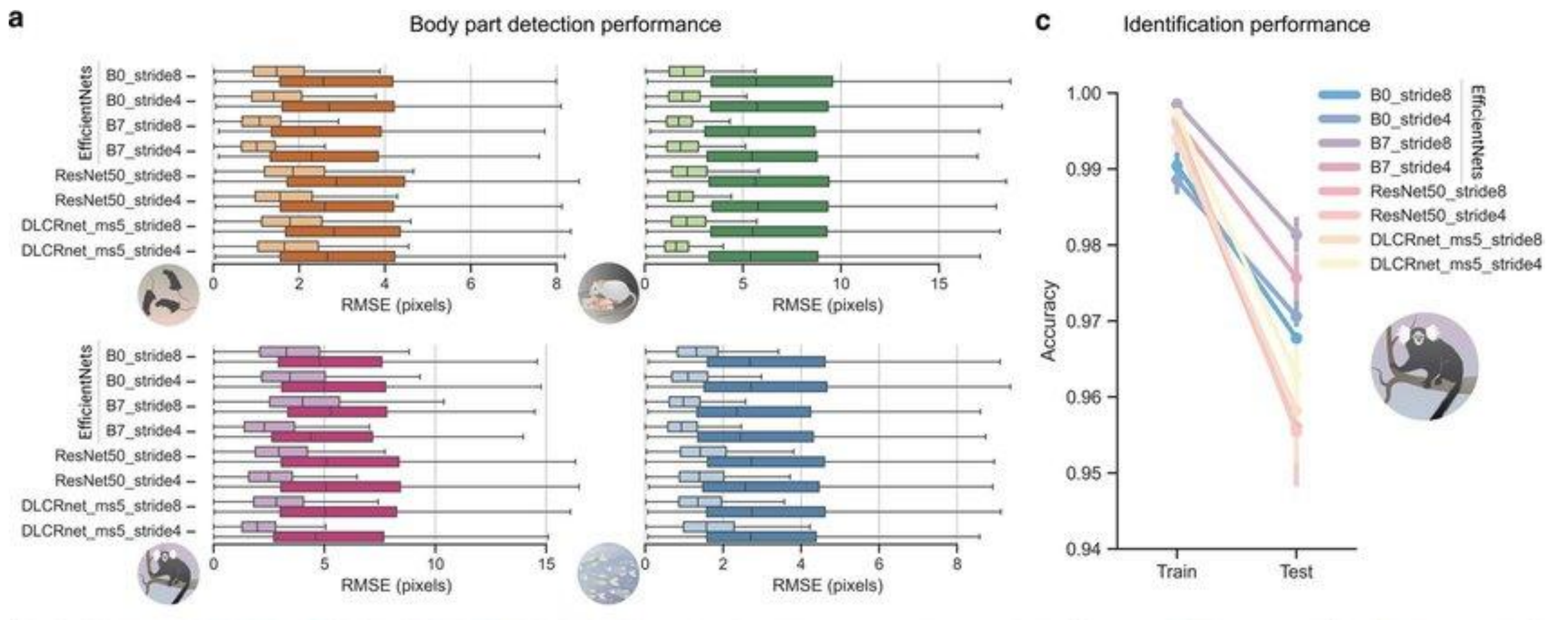
The following parameters modulate the data augmentation. During training, each image will be randomly rescaled within the range `[scale_jitter_lo, scale_jitter_up]` to augment training:

- `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.

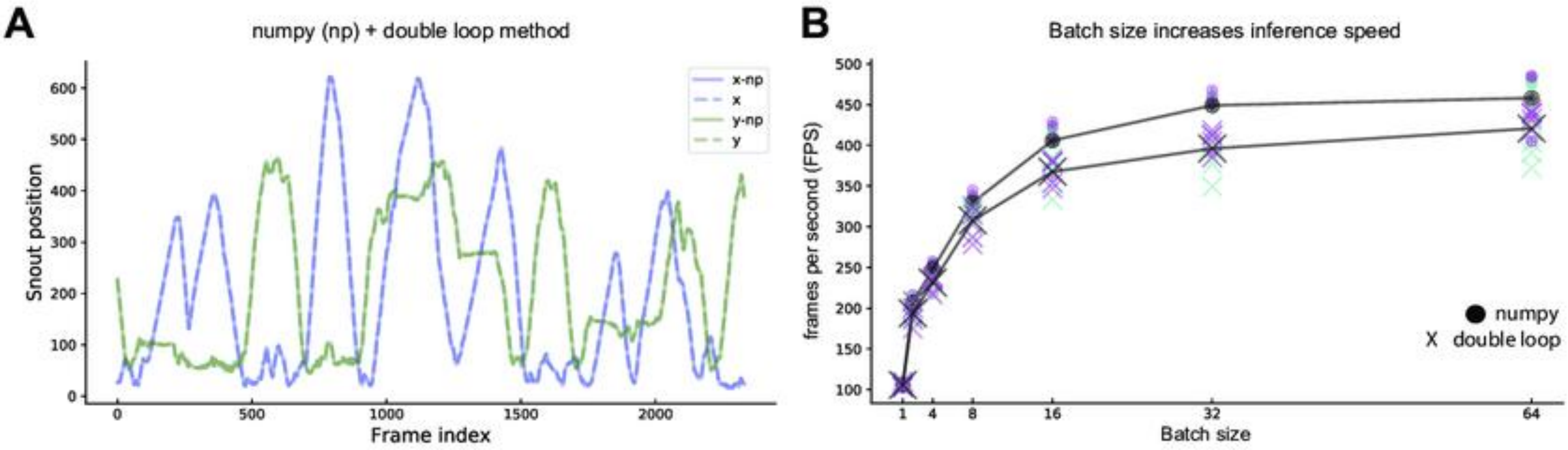
Nath et al. (2019). Nature Protocols.
https://www.nature.com/articles/s41596-019-0176-0/tables/1

Other type of training that affect performance?

- Batch size (higher the better, but may demand more power)
- Image size
- Hardware (GPU vs CPU)
- Training iterations



Batch size


A
numpy (np) + double loop method
x-np
x
y-np
y
Snout position
Frame index
B
Batch size increases inference speed
frames per second (FPS)
Batch size
numpy
X double loop

Mathis et al. (2019). Arxiv.

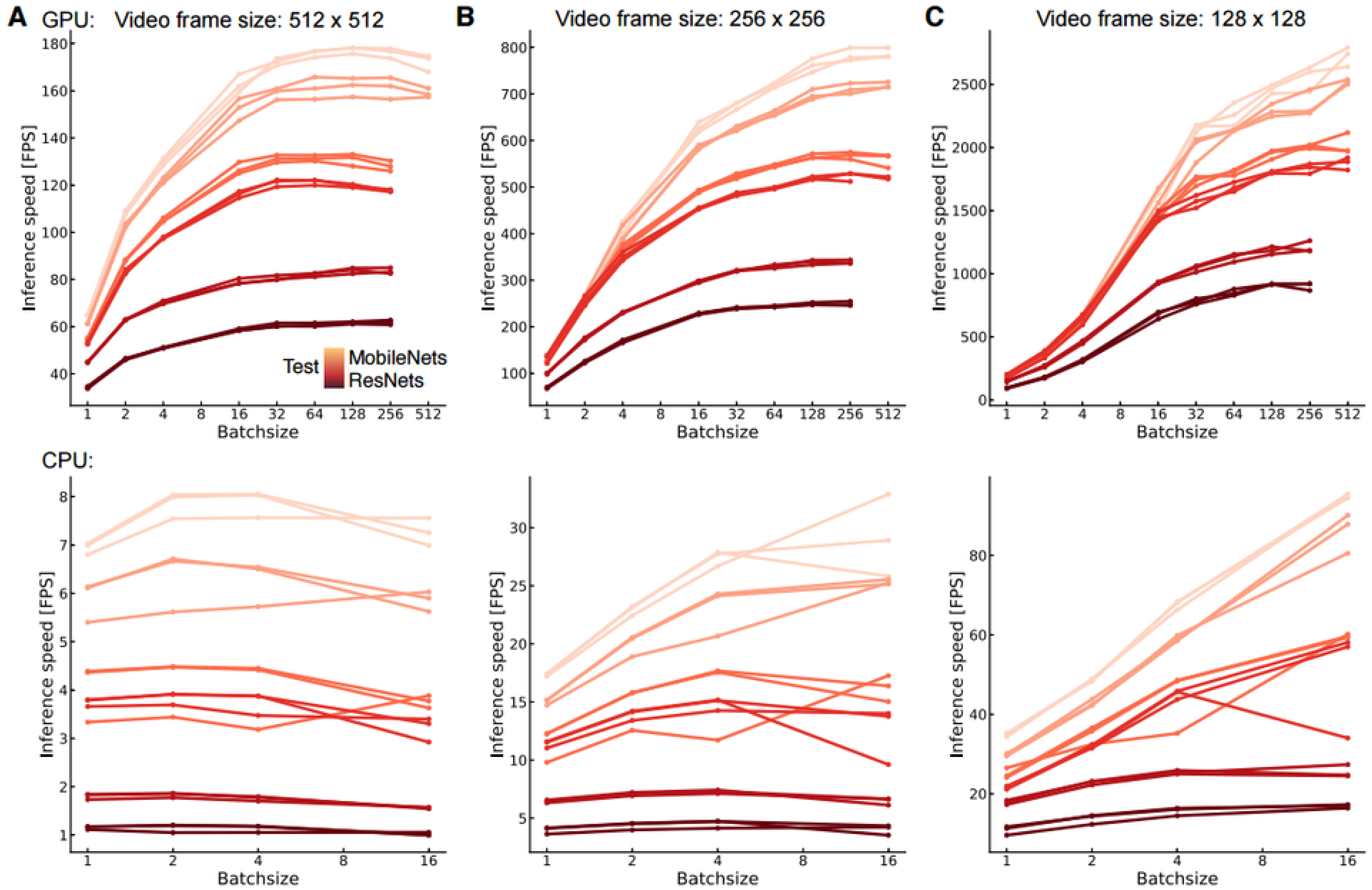
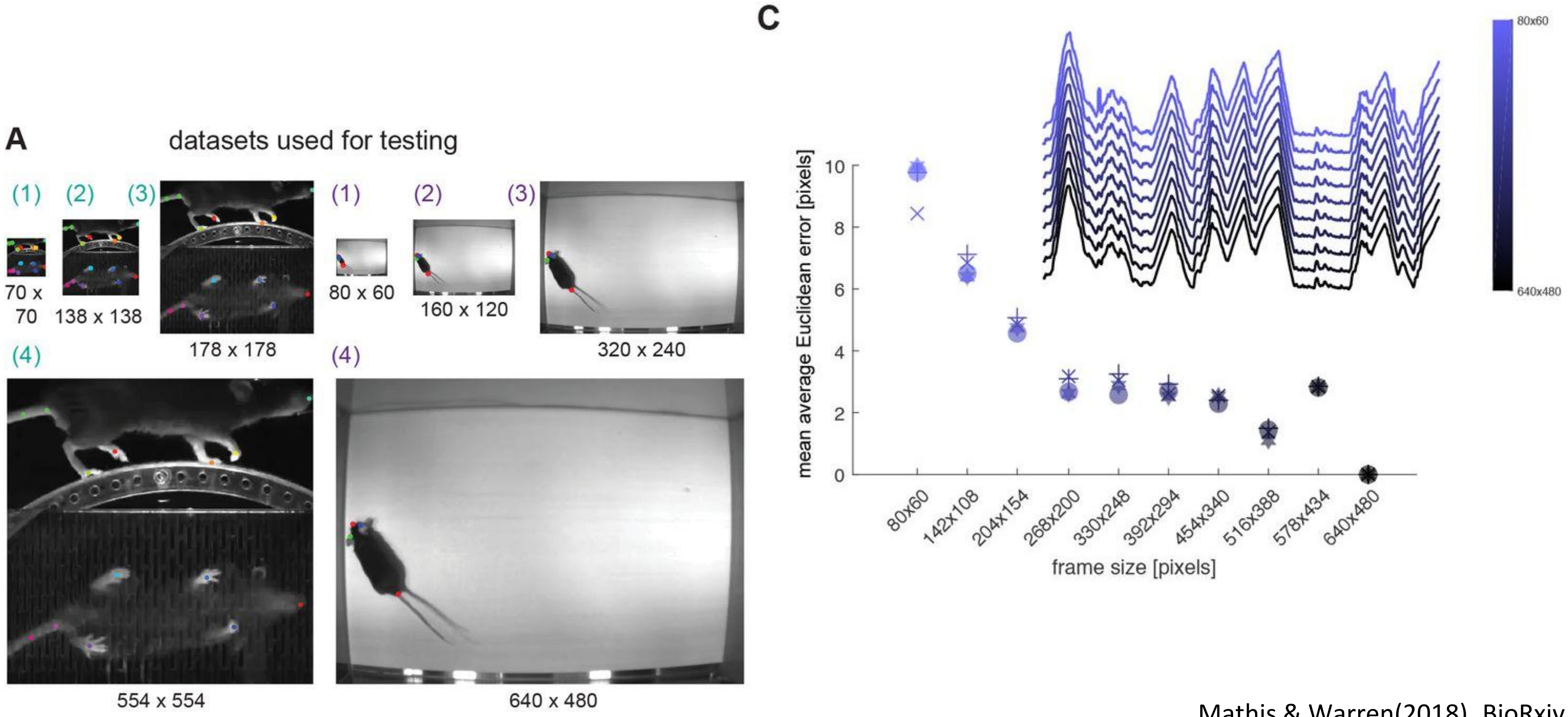


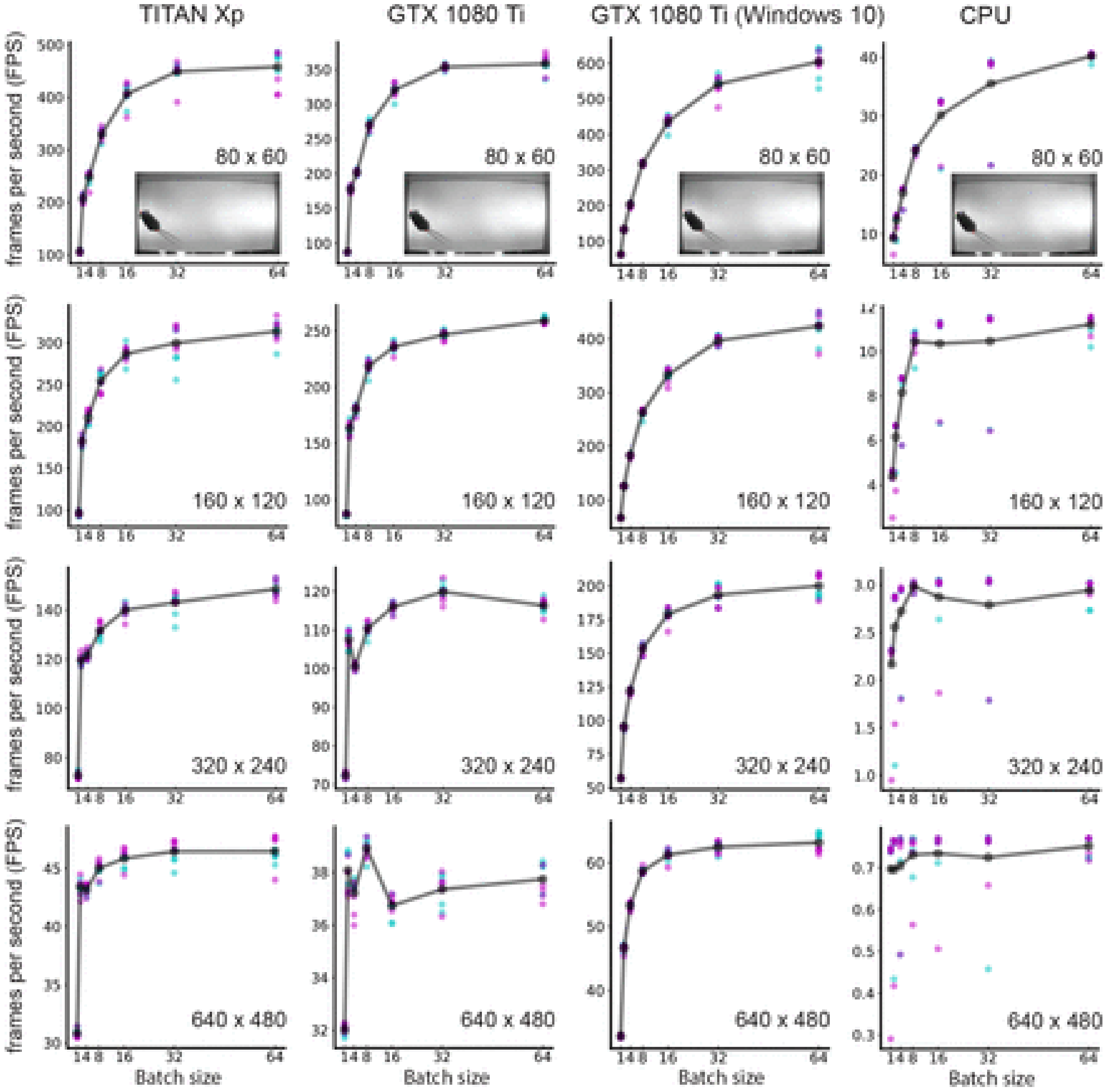
Image frame size vs error

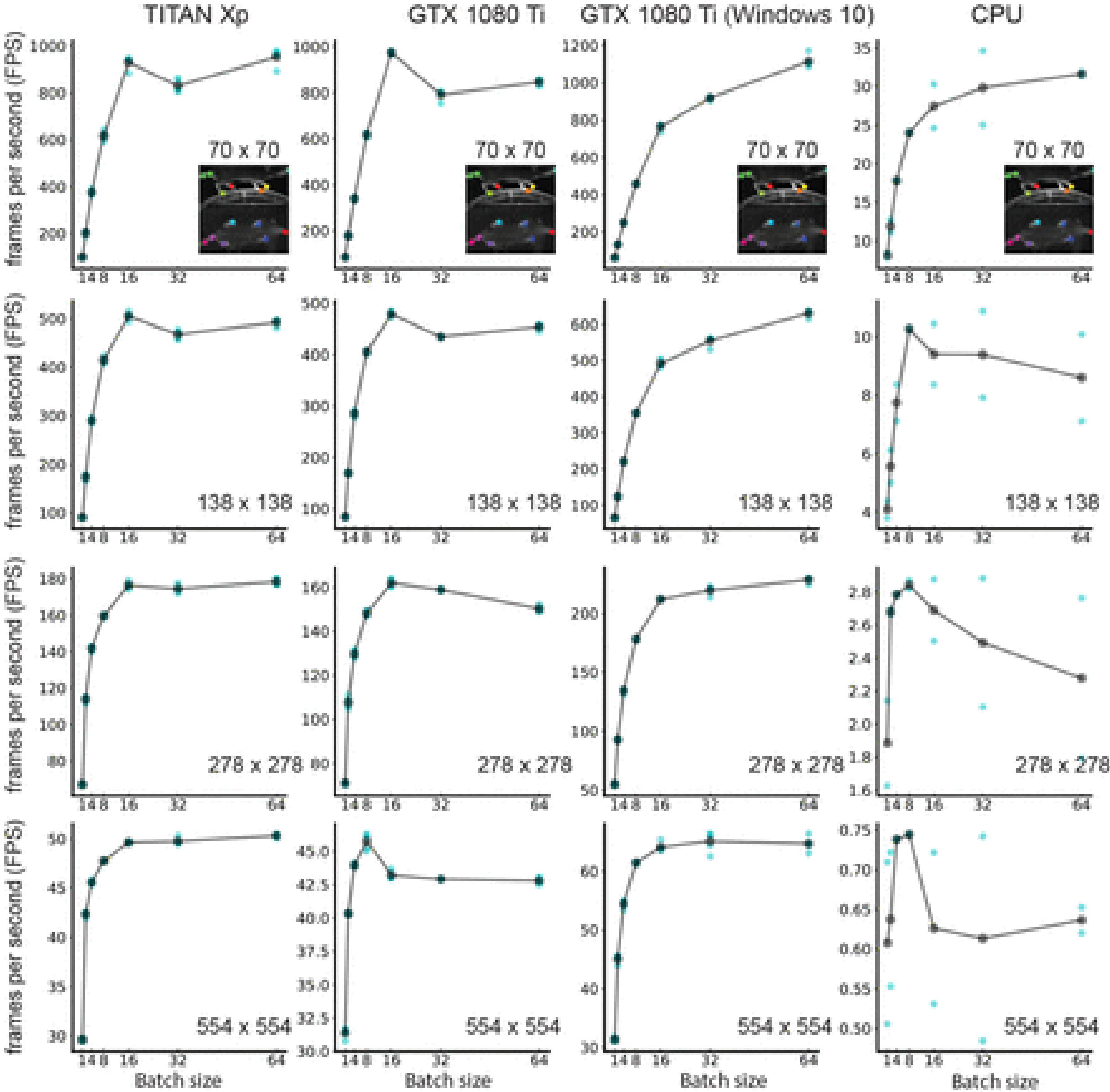


Mathis & Warren(2018). BioRxiv.

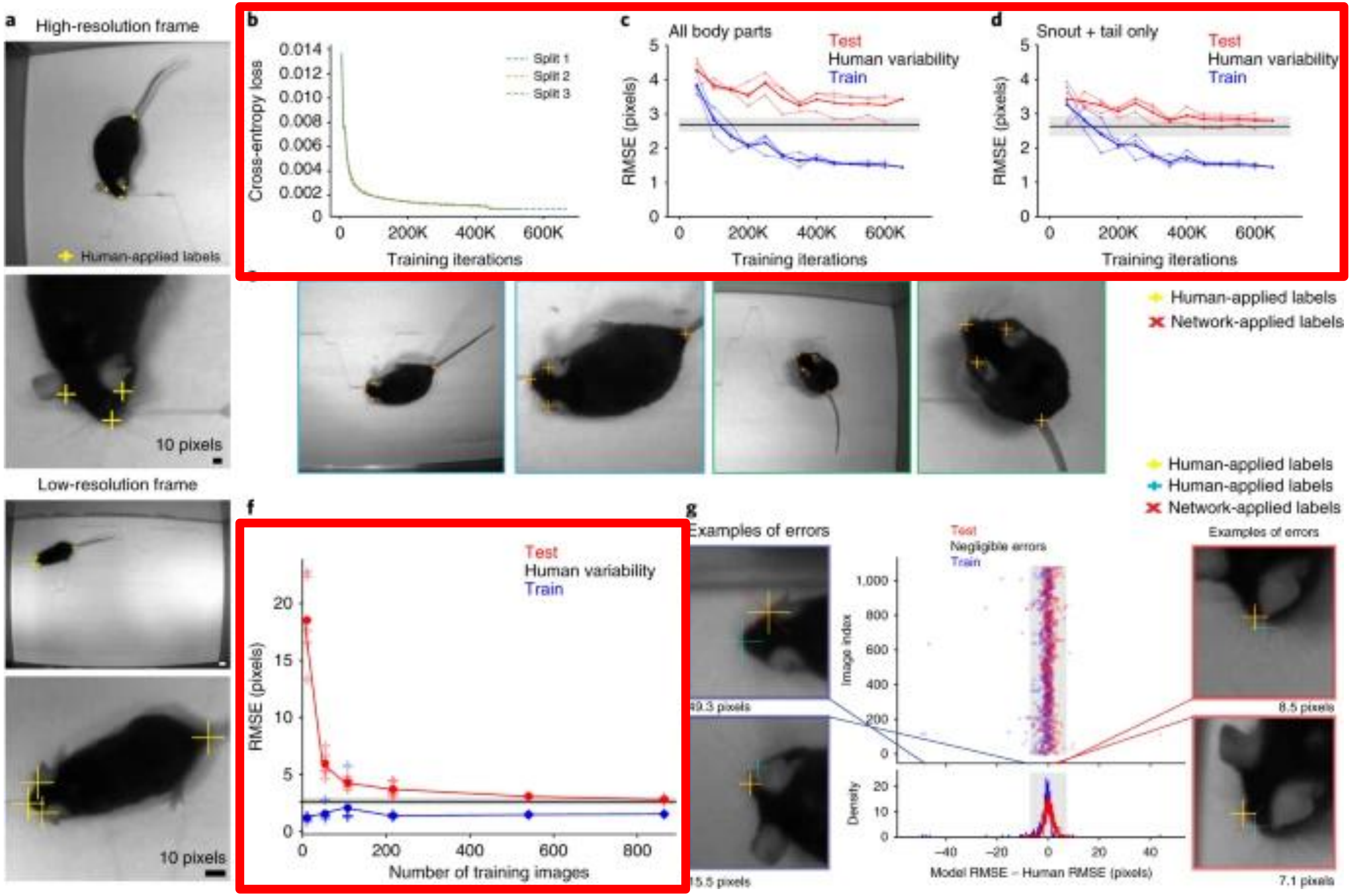
Mathis & Warren(2018). BioRxiv.

Hardware vs performance





Training iterations



Tips:

- Training speed mostly depends on frame size

  - GPU and CPU type

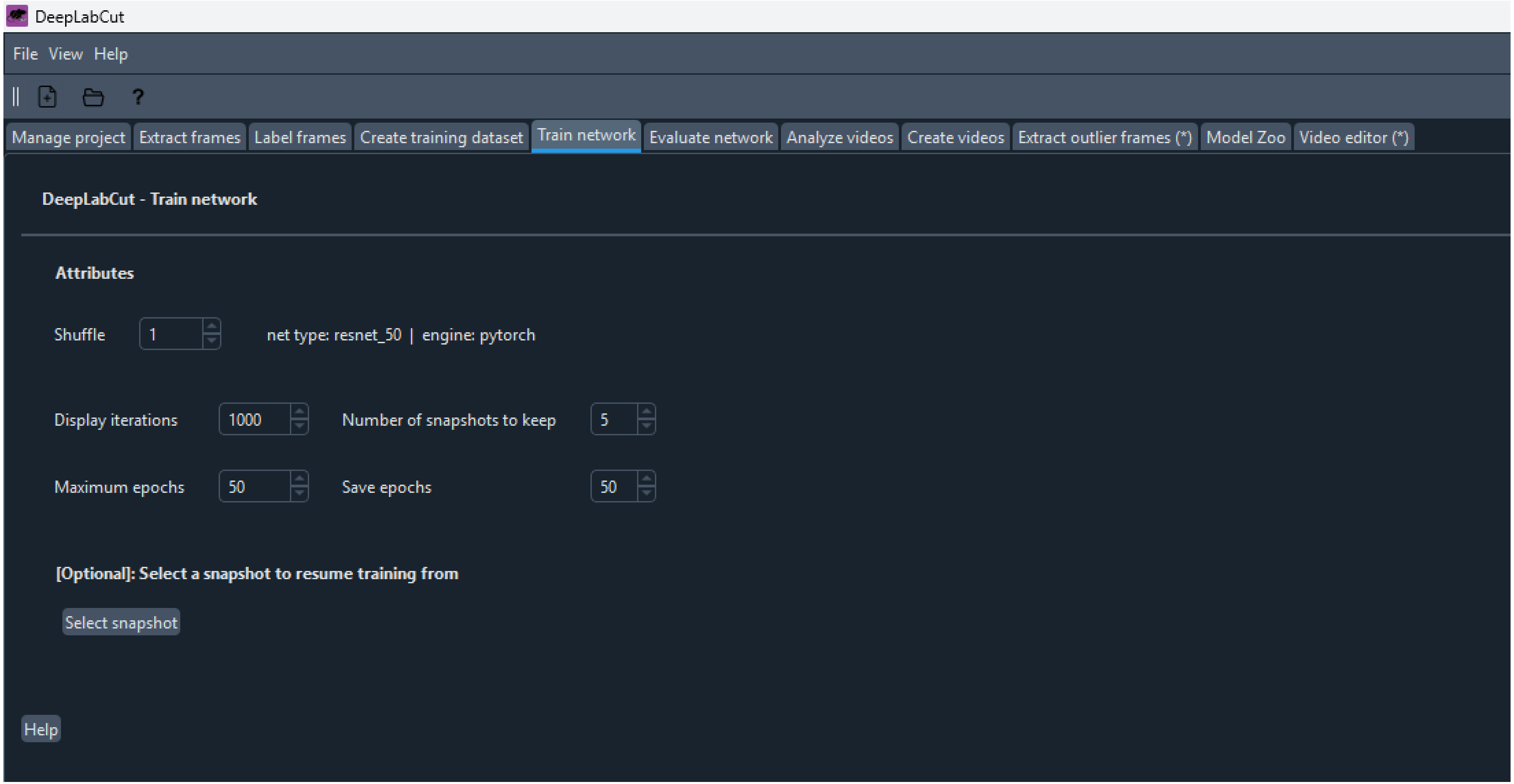
- Label at least 100–200 diverse frames

  - Complex task or multiple animals

- Monitor training loss regularly (should decrease over time)
- Use early stopping if loss plateaus or starts to increase
- Train with shuffle=1 to start

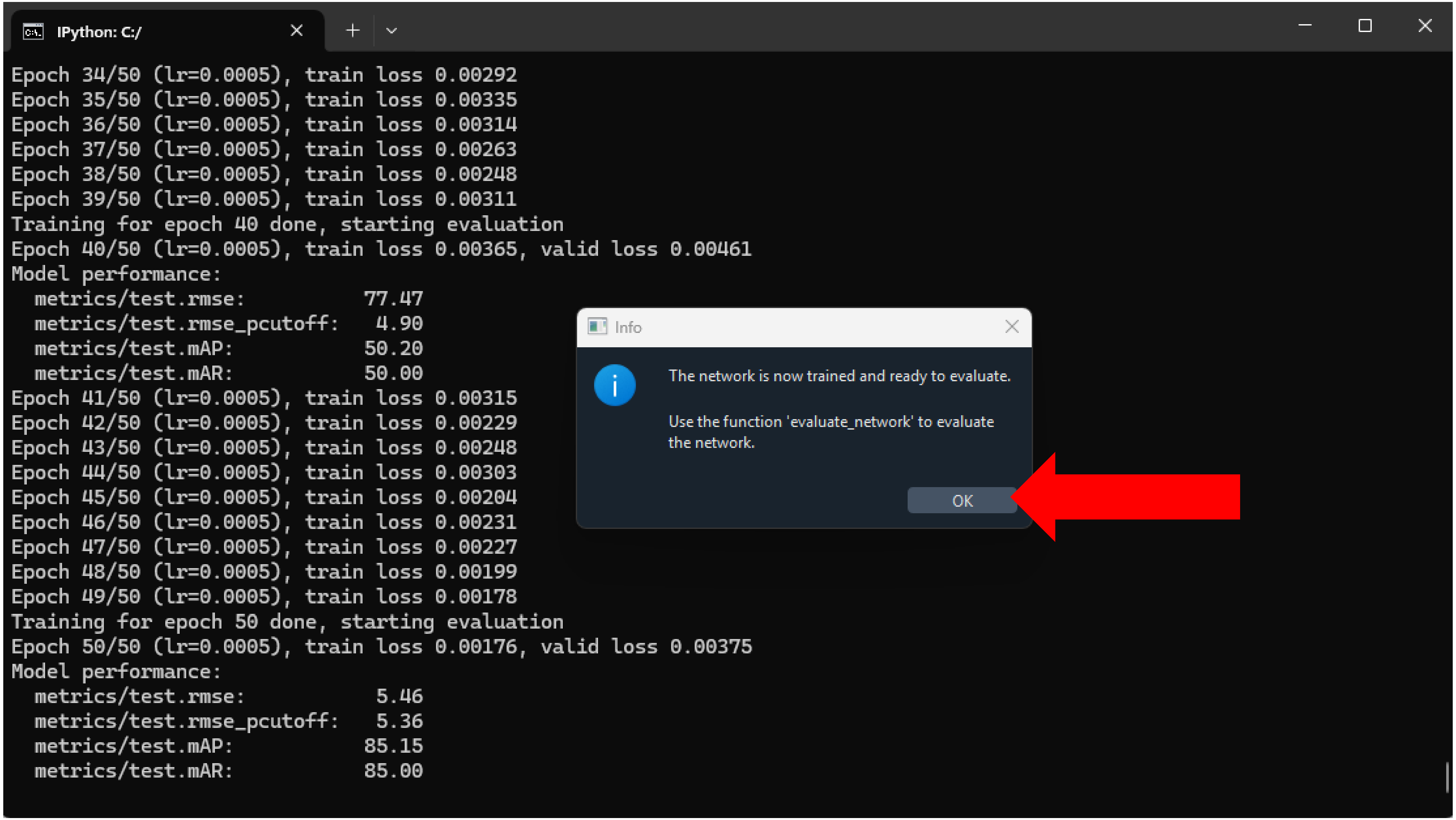
  - Can train additional shuffles for robustness

Next: try with your own dataset and videos to analyze!



```
IPython: C:/
[timm/resnet50_gn.a1h_in1k] Safe alternative available for 'pytorch_model.bin' (as 'model.safetensors'). Loading weights
 using safetensors.
Data Transforms:
  Training:   Compose([
  Affine(always_apply=False, p=0.5, interpolation=1, mask_interpolation=0, cval=0, mode=0, scale={'x': (0.5, 1.25), 'y':
 (0.5, 1.25)}, translate_percent=None, translate_px={'x': (0, 0), 'y': (0, 0)}, rotate=(-30, 30), fit_output=False, shea
r={'x': (0.0, 0.0), 'y': (0.0, 0.0)}, cval_mask=0, keep_ratio=True, rotate_method='largest_box'),
  PadIfNeeded(always_apply=True, p=1.0, min_height=448, min_width=448, pad_height_divisor=None, pad_width_divisor=None,
border_mode=0, value=None, mask_value=None),
  KeypointAwareCrop(always_apply=True, p=1.0, width=448, height=448, max_shift=0.1, crop_sampling='hybrid'),
  MotionBlur(always_apply=False, p=0.5, blur_limit=(3, 7), allow_shifted=True),
  GaussNoise(always_apply=False, p=0.5, var_limit=(0, 162.5625), per_channel=True, mean=0),
  Normalize(always_apply=False, p=1.0, mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], max_pixel_value=255.0),
], p=1.0, bbox_params={'format': 'coco', 'label_fields': ['bbox_labels'], 'min_area': 0.0, 'min_visibility': 0.0, 'min_w
idth': 0.0, 'min_height': 0.0, 'check_each_transform': True}, keypoint_params={'format': 'xy', 'label_fields': ['class_l
abels'], 'remove_invisible': False, 'angle_in_degrees': True, 'check_each_transform': True}, additional_targets={}, is_c
heck_shapes=True)
  Validation: Compose([
  Normalize(always_apply=False, p=1.0, mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], max_pixel_value=255.0),
], p=1.0, bbox_params={'format': 'coco', 'label_fields': ['bbox_labels'], 'min_area': 0.0, 'min_visibility': 0.0, 'min_w
idth': 0.0, 'min_height': 0.0, 'check_each_transform': True}, keypoint_params={'format': 'xy', 'label_fields': ['class_l
abels'], 'remove_invisible': False, 'angle_in_degrees': True, 'check_each_transform': True}, additional_targets={}, is_c
heck_shapes=True)
Using 25 images and 2 for testing

Starting pose model training...
--------------------------------------------------
Epoch 1/200 (lr=0.0005), train loss 0.01680
Epoch 2/200 (lr=0.0005), train loss 0.01519
```


IPython: C:/
Epoch 34/50 (lr=0.0005), train loss 0.00292
Epoch 35/50 (lr=0.0005), train loss 0.00335
Epoch 36/50 (lr=0.0005), train loss 0.00314
Epoch 37/50 (lr=0.0005), train loss 0.00263
Epoch 38/50 (lr=0.0005), train loss 0.00248
Epoch 39/50 (lr=0.0005), train loss 0.00311
Training for epoch 40 done, starting evaluation
Epoch 40/50 (lr=0.0005), train loss 0.00365, valid loss 0.00461
Model performance:
metrics/test.rmse: 77.47
metrics/test.rmse_pcutoff: 4.90
metrics/test.mAP: 50.20
metrics/test.mAR: 50.00
Epoch 41/50 (lr=0.0005), train loss 0.00315
Epoch 42/50 (lr=0.0005), train loss 0.00229
Epoch 43/50 (lr=0.0005), train loss 0.00248
Epoch 44/50 (lr=0.0005), train loss 0.00303
Epoch 45/50 (lr=0.0005), train loss 0.00204
Epoch 46/50 (lr=0.0005), train loss 0.00231
Epoch 47/50 (lr=0.0005), train loss 0.00227
Epoch 48/50 (lr=0.0005), train loss 0.00199
Epoch 49/50 (lr=0.0005), train loss 0.00178
Training for epoch 50 done, starting evaluation
Epoch 50/50 (lr=0.0005), train loss 0.00176, valid loss 0.00375
Model performance:
metrics/test.rmse: 5.46
metrics/test.rmse_pcutoff: 5.36
metrics/test.mAP: 85.15
metrics/test.mAR: 85.00
Info
The network is now trained and ready to evaluate.
Use the function 'evaluate_network' to evaluate the network.
OK

With multi-animal project (hippo testing videos)?

IPython: C:/

```
Epoch 184/200 (lr=1e-05), train loss 0.00257
Epoch 185/200 (lr=1e-05), train loss 0.00246
Epoch 186/200 (lr=1e-05), train loss 0.00228
Epoch 187/200 (lr=1e-05), train loss 0.00246
Epoch 188/200 (lr=1e-05), train loss 0.00245
Epoch 189/200 (lr=1e-05), train loss 0.00269
Training for epoch 190 done, starting evaluation
Epoch 190/200 (lr=1e-05), train loss 0.00226, valid loss 0.00446
Model performance:
  metrics/test.rmse:          3.96
  metrics/test.rmse_pcutoff:  3.96
  metrics/test.mAP:          88.32
  metrics/test.mAR:          90.00
Epoch 191/200 (lr=1e-05), train loss 0.00229
Epoch 192/200 (lr=1e-05), train loss 0.00256
Epoch 193/200 (lr=1e-05), train loss 0.00251
Epoch 194/200 (lr=1e-05), train loss 0.00243
Epoch 195/200 (lr=1e-05), train loss 0.00235
Epoch 196/200 (lr=1e-05), train loss 0.00246
Epoch 197/200 (lr=1e-05), train loss 0.00291
Epoch 198/200 (lr=1e-05), train loss 0.00250
Epoch 199/200 (lr=1e-05), train loss 0.00226
Training for epoch 200 done, starting evaluation
Epoch 200/200 (lr=1e-05), train loss 0.00285, valid loss 0.00448
Model performance:
  metrics/test.rmse:          3.95
  metrics/test.rmse_pcutoff:  3.95
  metrics/test.mAP:          88.32
  metrics/test.mAR:          90.00
```

Info

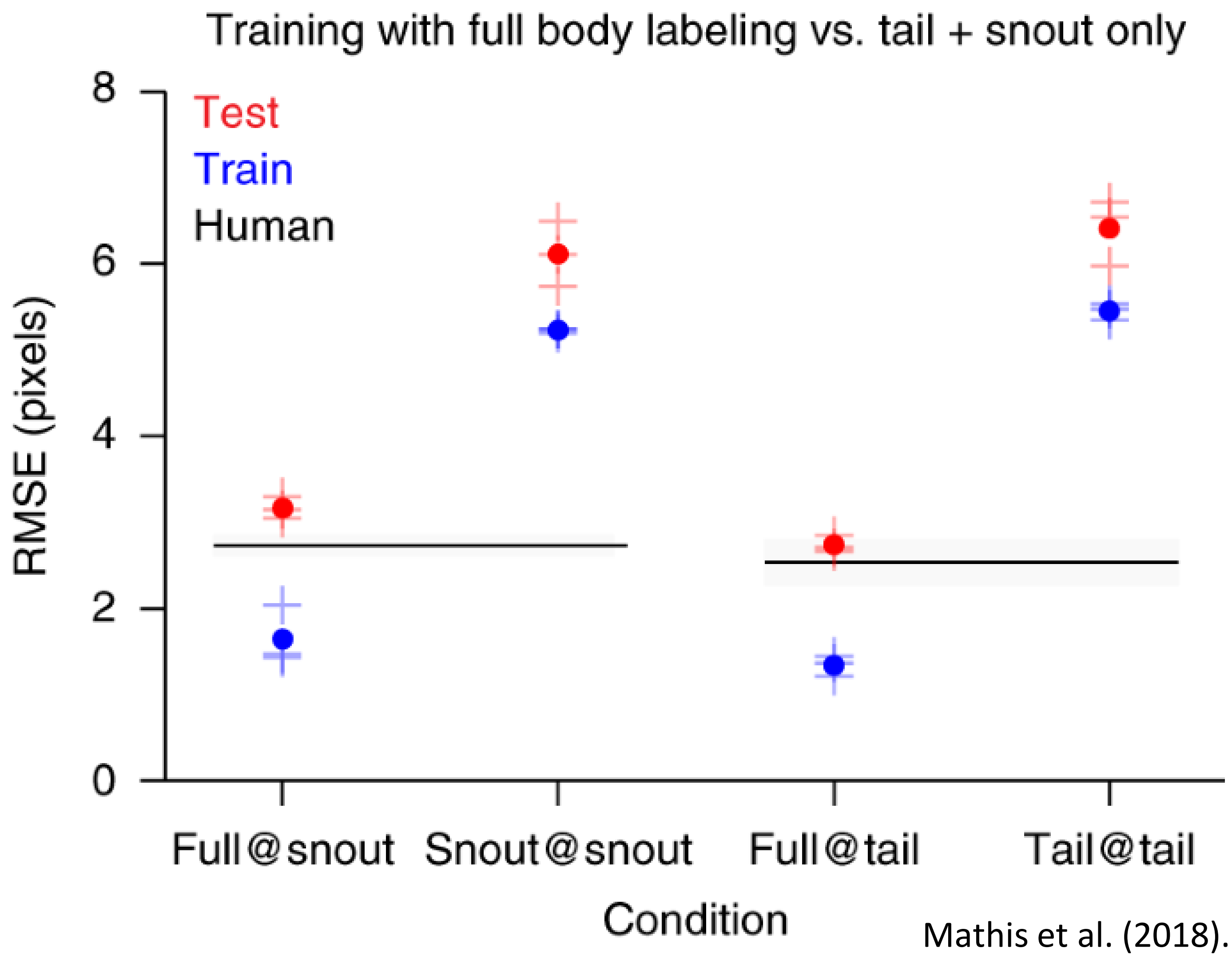
The network is now trained and ready to evaluate.

Use the function 'evaluate_network' to evaluate the network.

OK

Next step: **evaluation**

*Check labeling **RMSE** (Root Mean Square Error)!*



Mathis et al. (2018). Nature Neuroscience.