Figures and figure supplements

DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning

Jacob M Graving *et al*

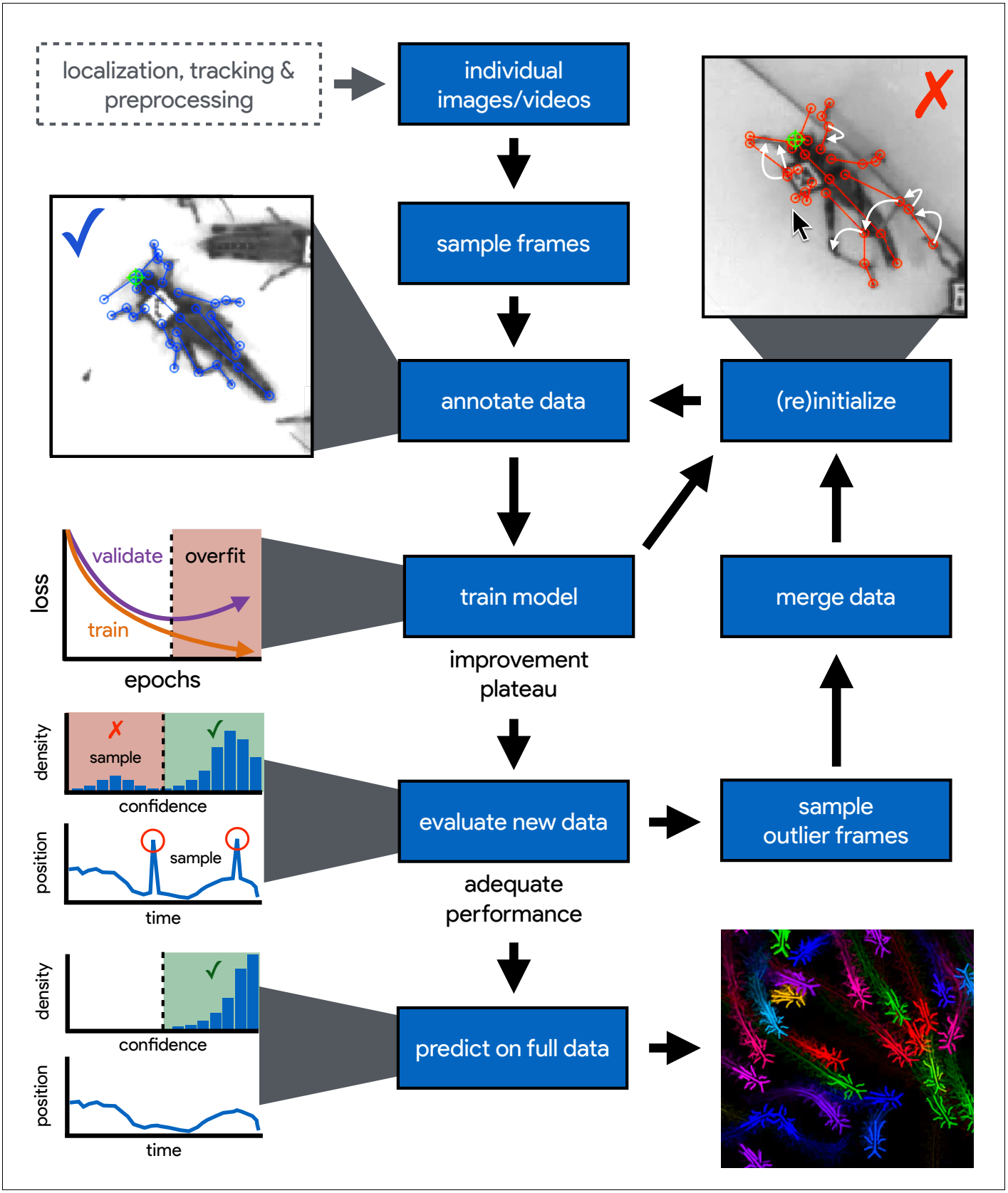


Figure 1. An illustration of the workflow for DeepPoseKit. Multi-individual images are localized, tracked, and preprocessed into individual images, which is not required for single-individual image datasets. An initial image set is sampled, annotated, and then iteratively updated using the active

Figure 1 continued on next page

Figure 1 continued

learning approach described by ***Pereira et al. (2019)*** (see Appendix 3). As annotations are made, the model is trained (***Figure 2***) with the current training set and keypoint locations are initialized for unannotated data to reduce the difficulty of further annotations. This is repeated until there is a noticeable improvement plateau for the initialized data—where the annotator is providing only minor corrections—and for the validation error when training the model (***Appendix 1—figure 4***). New data from the full dataset are evaluated with the model, and the training set is merged with new examples that are sampled based on the model's predictive performance, which can be assessed with techniques described by ***Mathis et al. (2018)*** and ***Nath et al. (2019)*** for identifying outlier frames and minimizing extreme prediction errors—shown here as the distribution of confidence scores predicted by the model and predicted body part positions with large temporal derivatives, indicating extreme errors. This process is repeated as necessary until performance is adequate when evaluating new data. The pose estimation model can then be used to make predictions for the full data set, and the data can be used for further analysis.

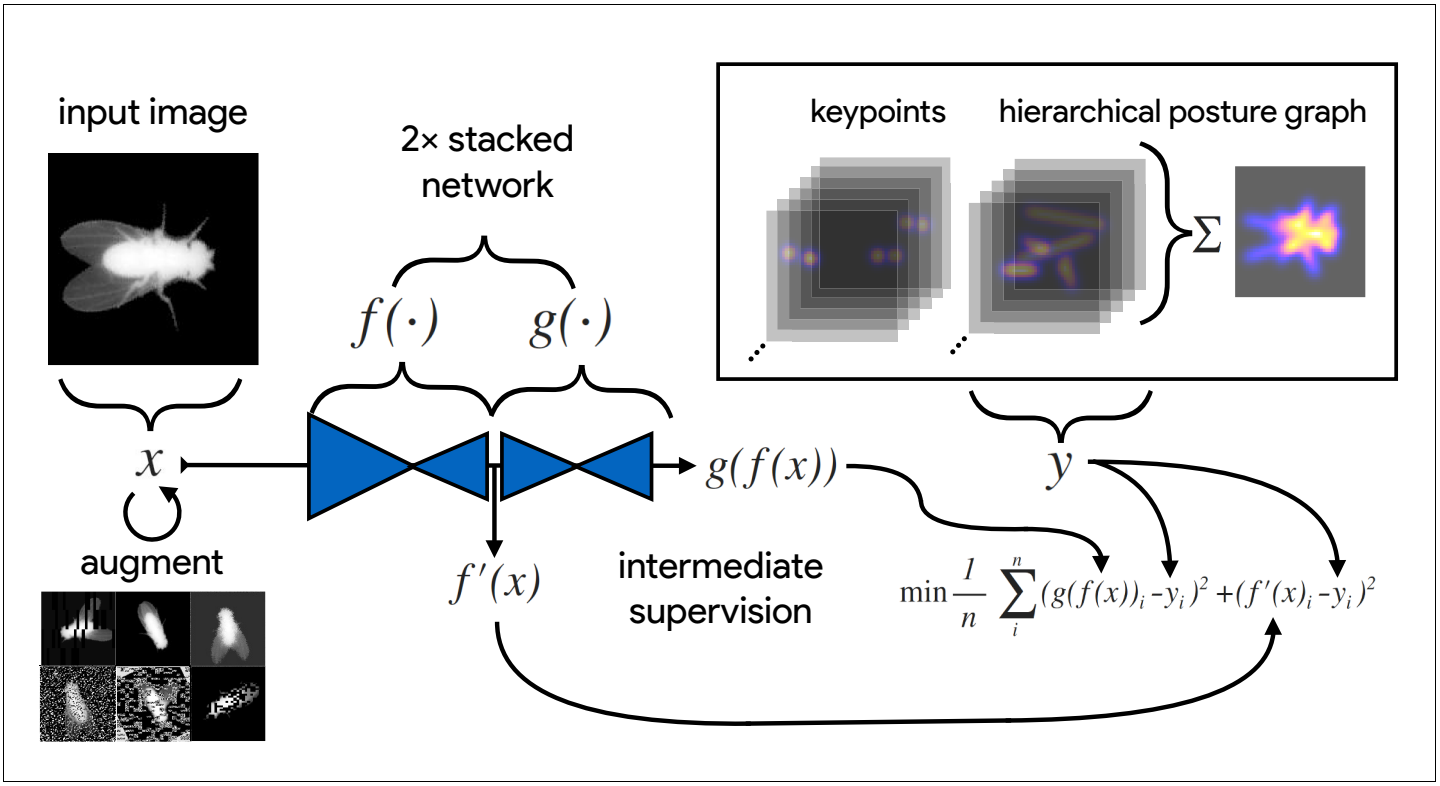


Figure 2. An illustration of the model training process for our Stacked DenseNet model in DeepPoseKit (see Appendix 2 for details about training models). Input images x (top-left) are augmented (bottom-left) with various spatial transformations (rotation, translation, scale, etc.) followed by noise transformations (dropout, additive noise, blurring, contrast, etc.) to improve the robustness and generalization of the model. The ground truth annotations are then transformed with matching spatial augmentations (not shown for the sake of clarity) and used to draw the confidence maps y for the keypoints and hierarchical posture graph (top-right). The images x are then passed through the network to produce a multidimensional array $g(f(x))$—a stack of images corresponding to the keypoint and posture graph confidence maps for the ground truth y. Mean squared error between the outputs for both networks $g(f(x))$ and $f'(x)$ and the ground truth data y is then minimized (bottom-right), where $f'(x)$ indicates a subset of the output from $f(x)$—only those feature maps being optimized to reproduce the confidence maps for the purpose of intermediate supervision (Appendix 5). The loss function is minimized until the validation loss stops improving—indicating that the model has converged or is starting to overfit to the training data.

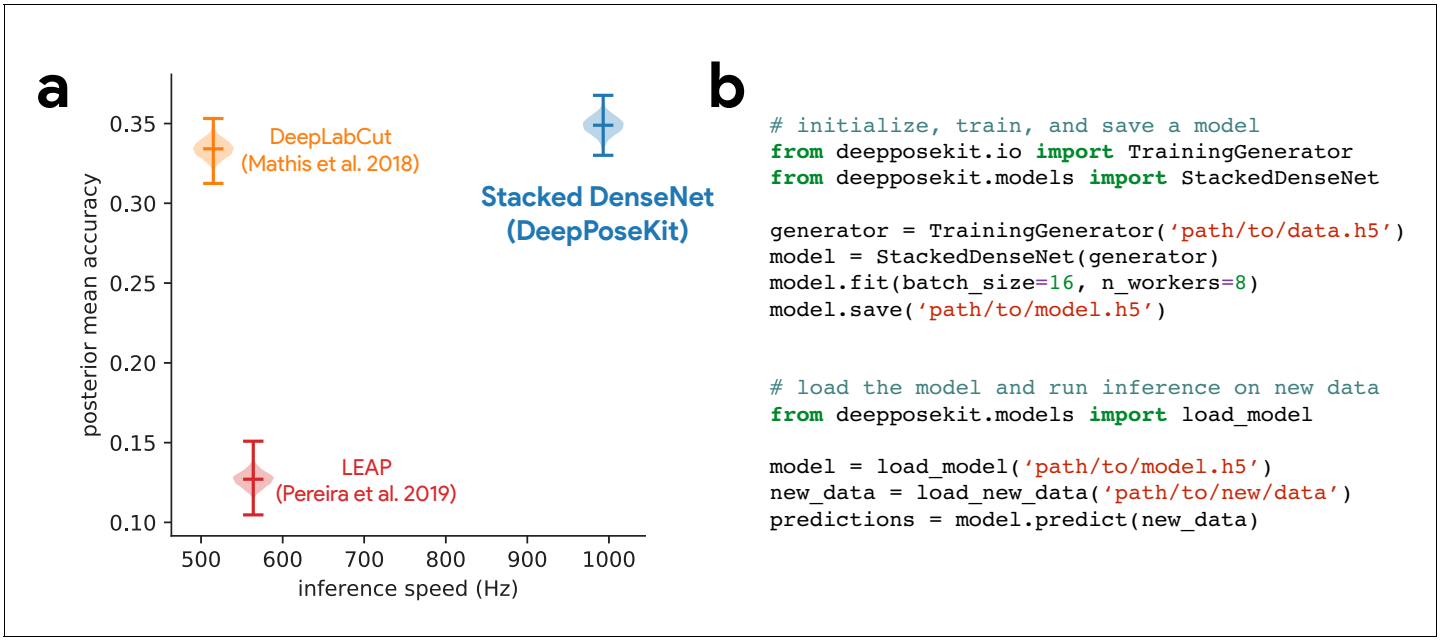


Figure 3. DeepPoseKit is fast, accurate, and easy-to-use. Our Stacked DenseNet model estimates posture at approximately 2×—or greater—the speed of the LEAP model (***Pereira et al., 2019***) and the DeepLabCut model (***Mathis et al., 2018***) while also achieving similar accuracy to the DeepLabCut model (***Mathis et al., 2018***)—shown here as mean accuracy $(1 + Euclideanerror)^{-1}$ for our most challenging dataset of multiple interacting Grévy's zebras (*E. grevyi*) recorded in the wild (**a**). See ***Figure 3—figure supplement 1*** for further details. Our software interface is designed to be straightforward but flexible. We include many options for expert users to customize model training with sensible default settings to make pose estimation as easy as possible for beginners. For example, training a model and running inference on new data requires writing only a few lines of code and specifying some basic settings (**b**).

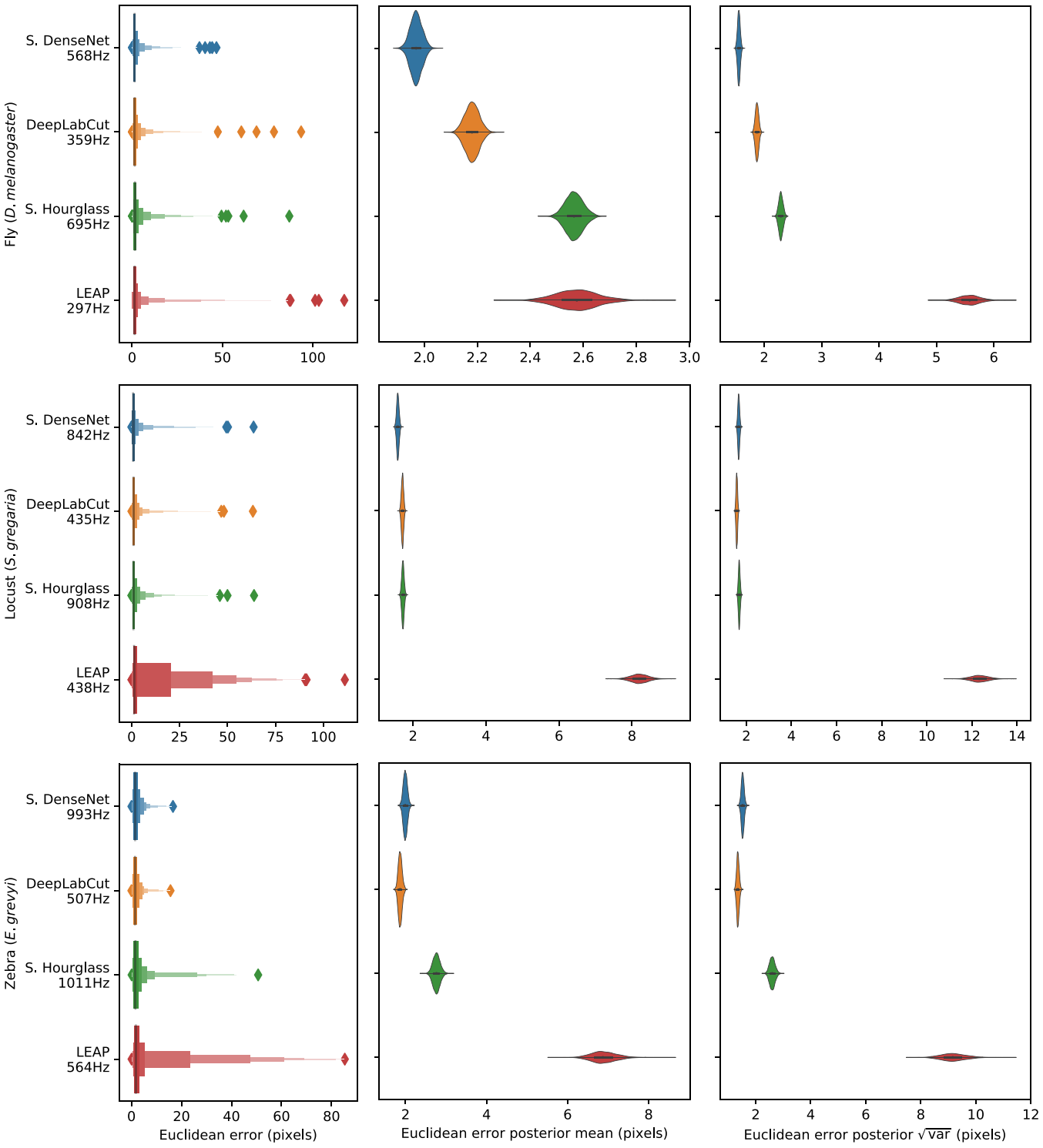


Figure 3—figure supplement 1. Euclidean error distributions for each model across our three datasets. Letter-value plots (left) show the raw error distributions for each model. Violinplots of the posterior distributions for the mean and variance (right) show statistical differences between the error distributions. Overall the LEAP model (***Pereira et al., 2019***) was the worst performer on every dataset in terms of both mean and variance. Our Stacked DenseNet model was the best performer for the fly dataset, while our Stacked DenseNet model and the DeepLabCut model (***Mathis et al., 2018***) both

Figure 3—figure supplement 1 continued on next page

Figure 3—figure supplement 1 continued

performed equally well on the locust and zebra datasets. The posteriors for the DeepLabCut model (***Mathis et al., 2018***) and our Stacked DenseNet model are highly overlapping for these datasets, which suggests they are not statistically discernible from one another. Our Stacked Hourglass model (***Newell et al., 2016***) performed equally to the DeepLabCut model (***Mathis et al., 2018***) and our Stacked DenseNet model for the locust dataset but performed slightly worse for the fly and zebra datasets.

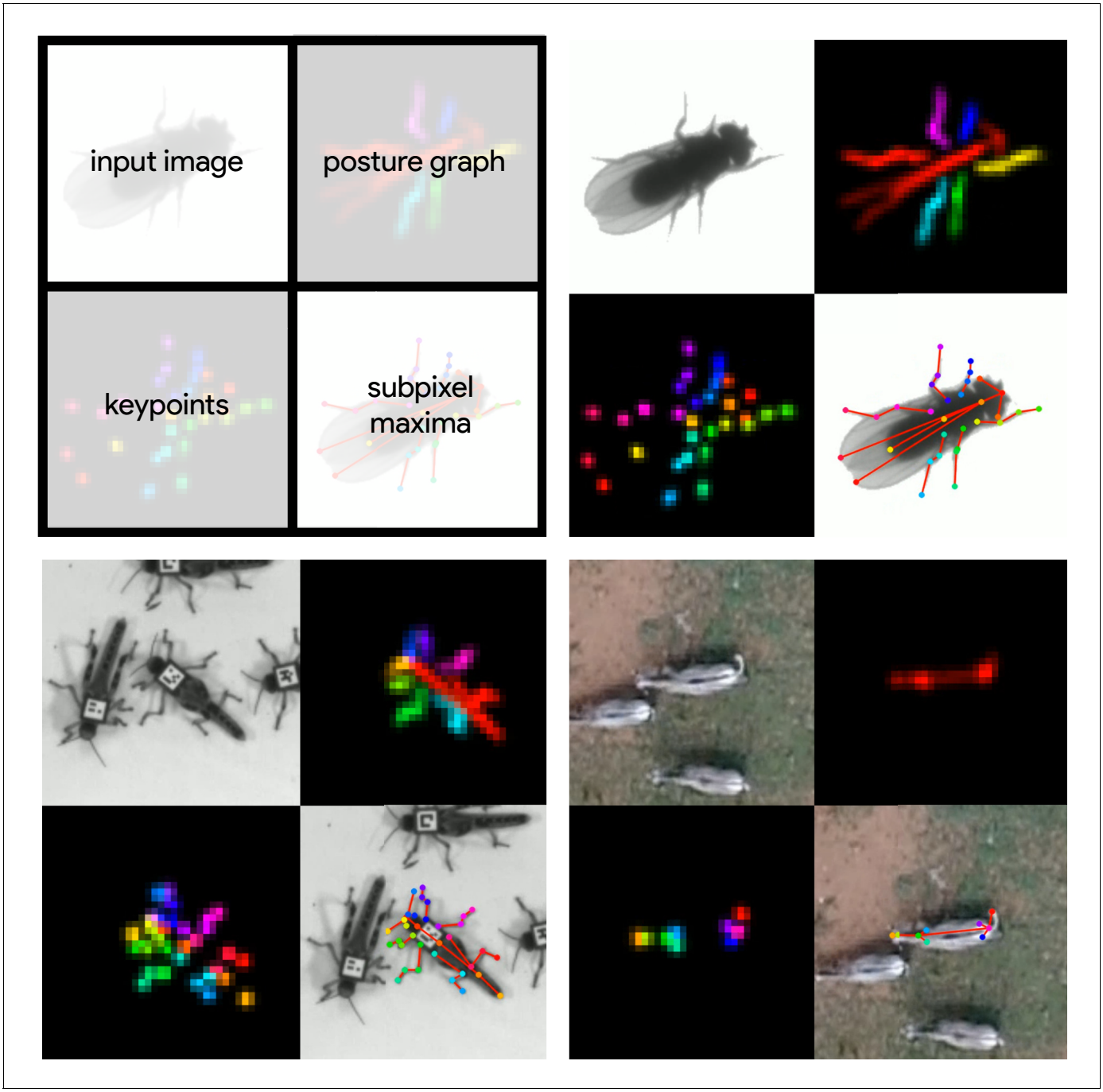
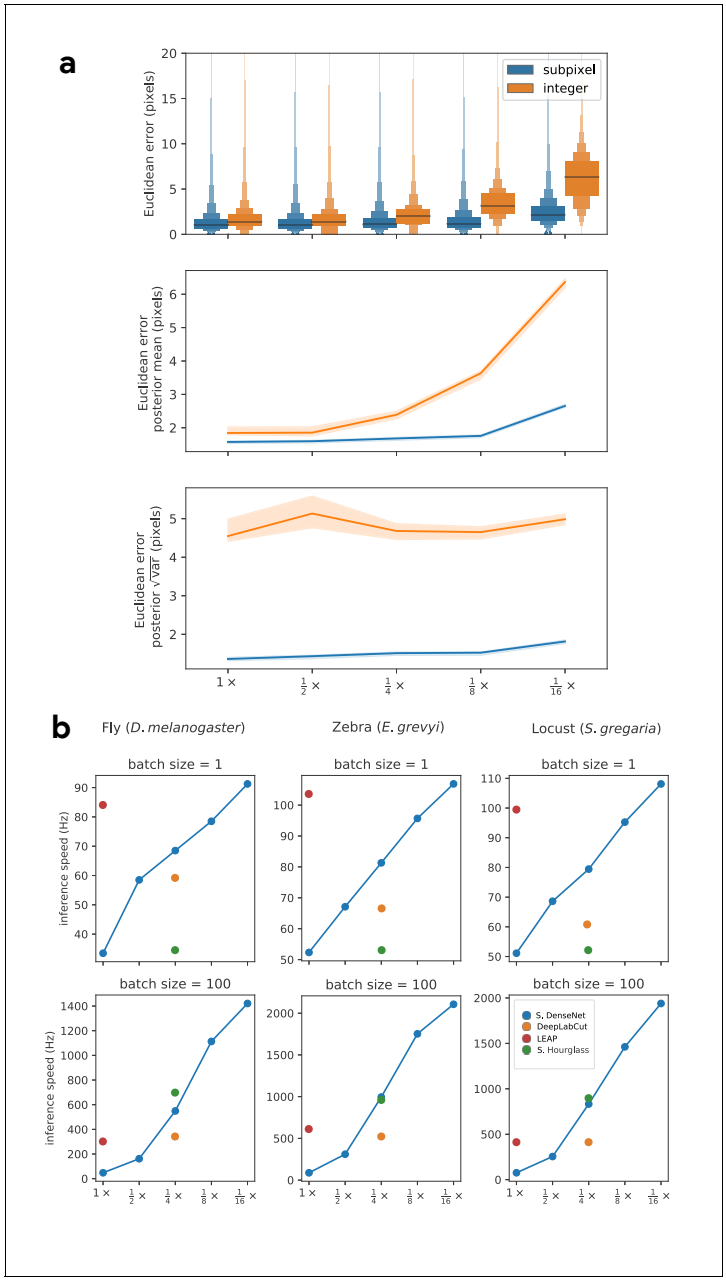
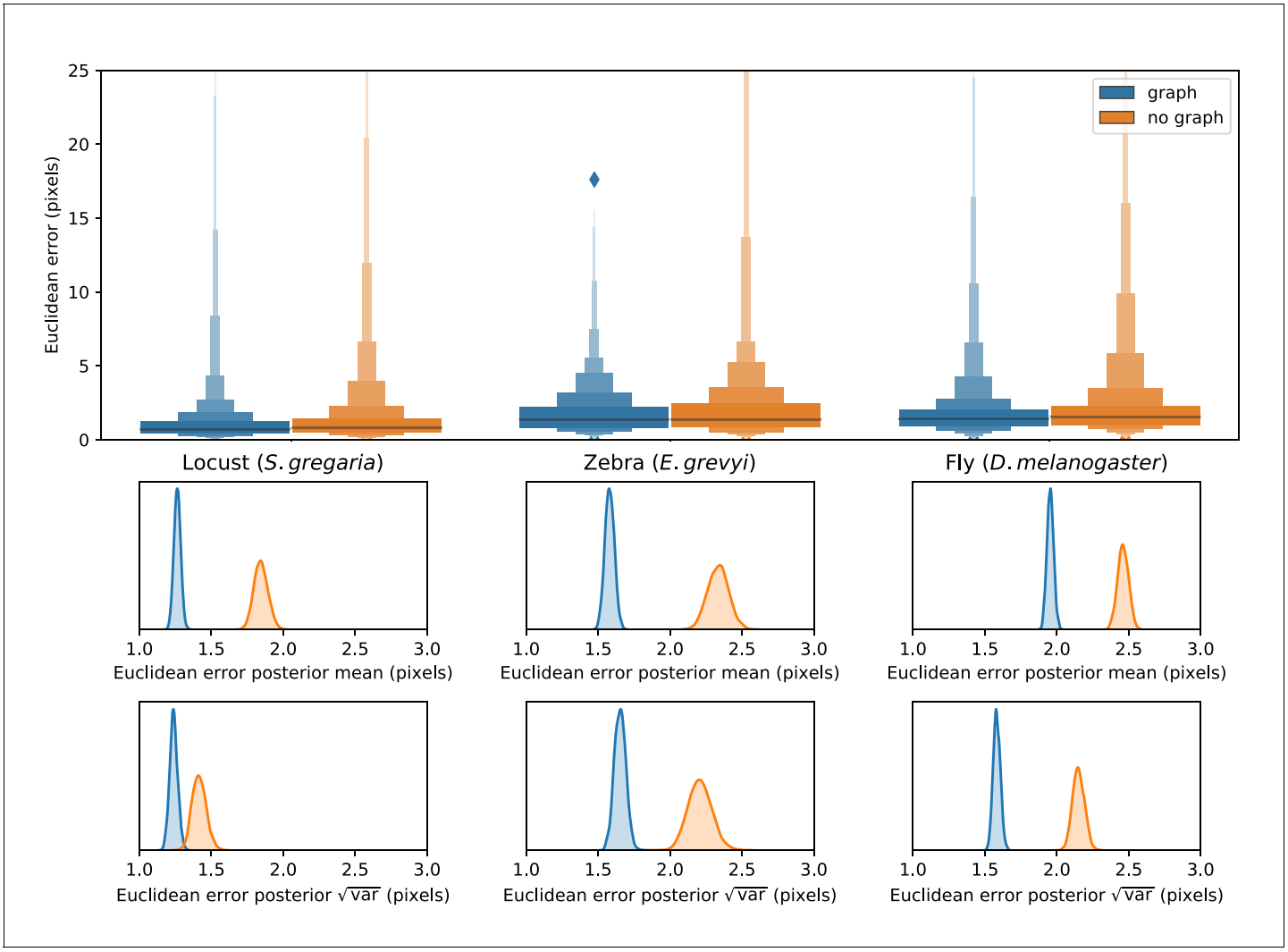


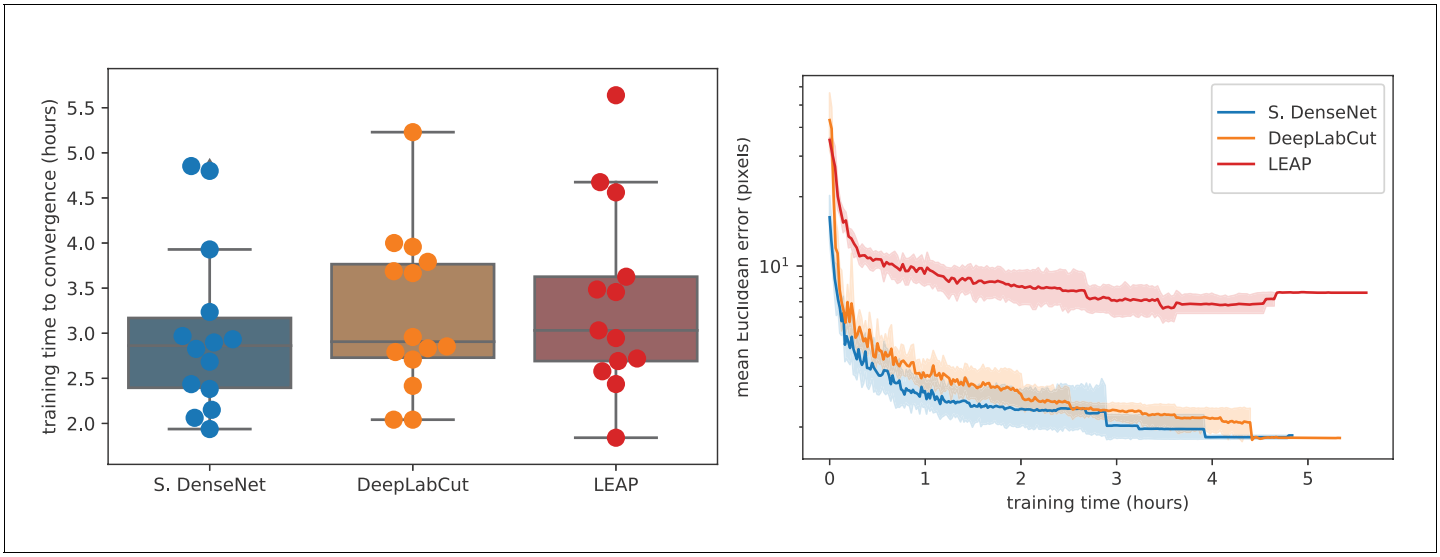
Figure 4. Datasets used for evaluation. A visualization of the datasets we used to evaluate our methods (***Table 1***). For each dataset, confidence maps for the keypoints (bottom-left) and posture graph (top-right) are illustrated using different colors for each map. These outputs are from our Stacked DenseNet model at $\frac{1}{4}\times$ resolution.



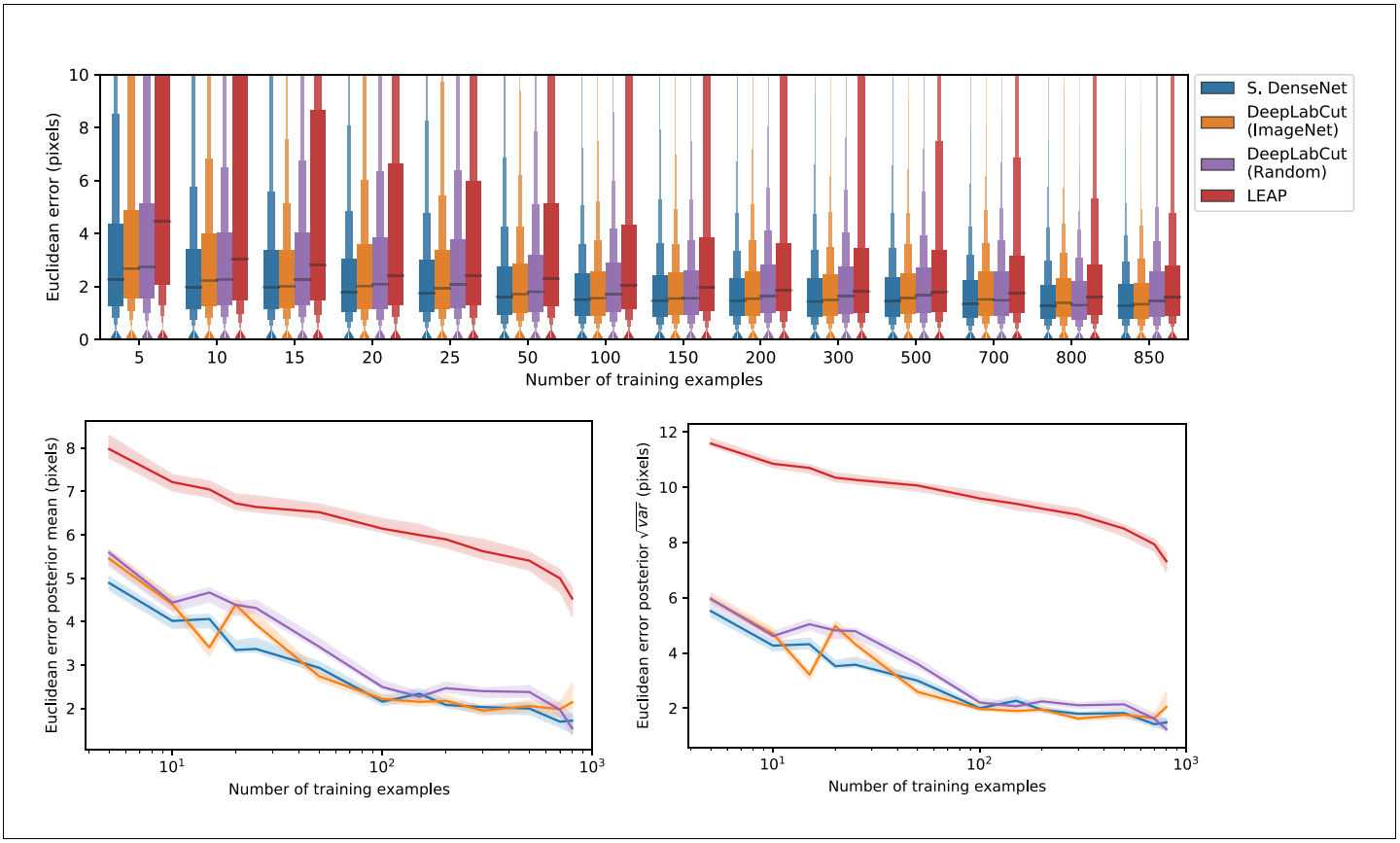
Appendix 1—figure 1. Our subpixel maxima algorithm increases speed without decreasing accuracy. Prediction accuracy on the fly dataset is maintained across downsampling configurations (**a**). Letter-value plots (a-top) show the raw error distributions for each configuration. Visualizations of the credible intervals (99% highest-density region) of the posterior distributions for the mean and variance (a-bottom) illustrate statistical differences between the error distributions, where using subpixel maxima decreases both the mean and variance of the error distribution. Inference speed is fast and can be run in real-time on single images (batch size = 1) at ~30–110 Hz or offline (batch size = 100) upwards of 1000 Hz (**b**). Plots show the inference speeds for our Stacked DenseNet model across downsampling configurations as well as the other models we tested for each of our datasets.



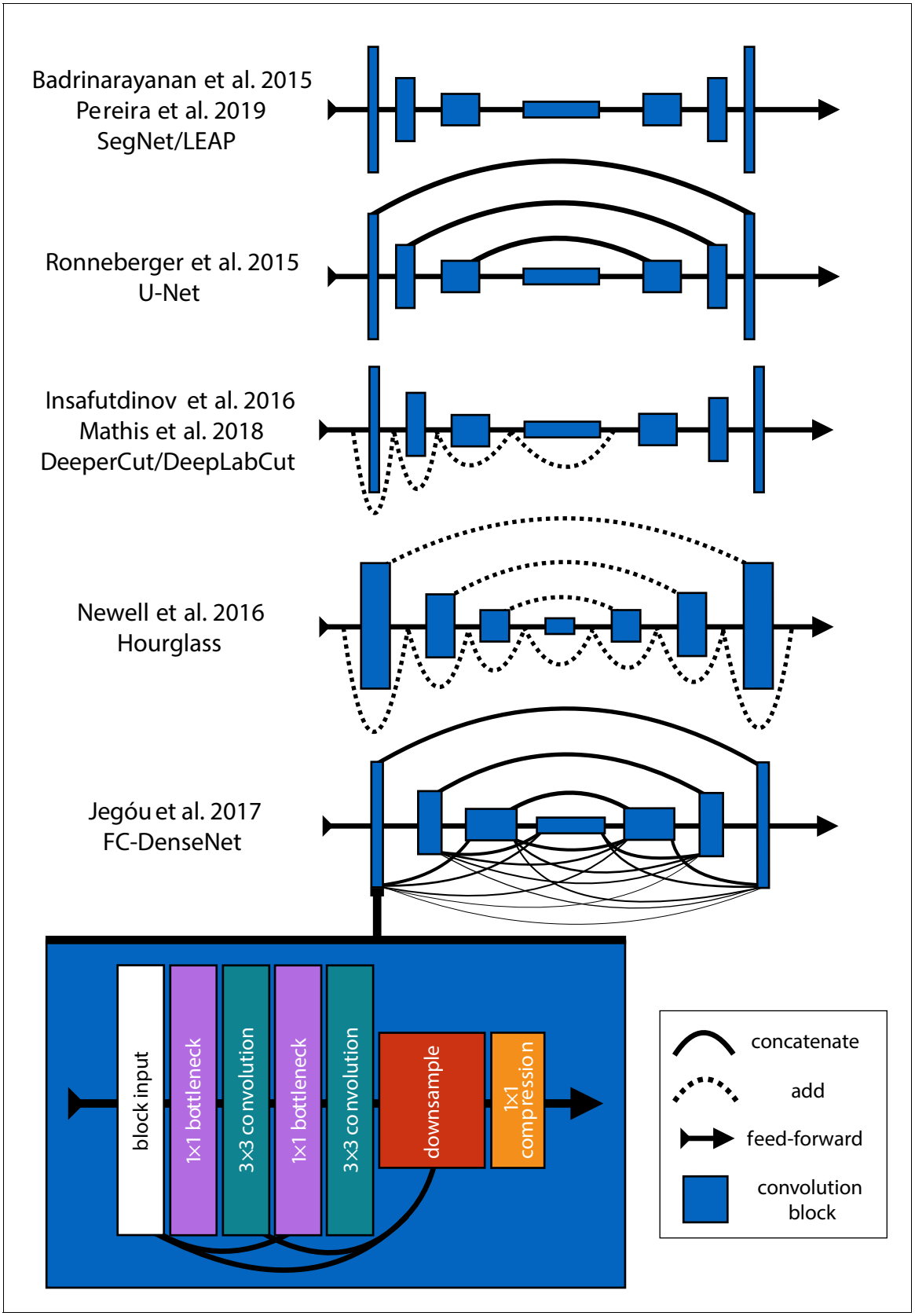
Appendix 1—figure 2. Predicting the multi-scale geometry of the posture graph reduces error. Letter-value plots (top) show the raw error distributions for each experiment. Visualizations of the posterior distributions for the mean and variance (bottom) show statistical differences between the error distributions. Predicting the posture graph decreases both the mean and variance of the error distribution.



Appendix 1—figure 3. Training time required for our Stacked DenseNet model, the DeepLabCut model (***Mathis et al., 2018***), and the LEAP model (***Pereira et al., 2019***) (n = 15 per model) using our zebra dataset. Boxplots and swarm plots (left) show the total training time to convergence (<0.001 improvement in validation loss for 50 epochs). Line plots (right) illustrate the Euclidean error of the validation set during training, where error bars show bootstrapped (n = 1000) 99% confidence intervals of the mean. Fully training models to convergence requires only a few hours of optimization (left) with reasonable accuracy reached after only 1 hr (right) for our Stacked DenseNet model.



Appendix 1—figure 4. A comparison of prediction accuracy with different numbers of training examples from our zebra dataset. The error distributions shown as letter-value plots (top) illustrate the Euclidean error for the remainder of the dataset not used for training—with a total of 900 labeled examples in the dataset. Line plots (bottom) show posterior credible intervals (99% highest-density region) for the mean and variance of the error distributions. We tested our Stacked DenseNet model; the DeepLabCut model (***Mathis et al., 2018***) with transfer learning—that is with weights pretrained on ImageNet (***Deng et al., 2009***); the same model without transfer learning—that is with randomly-initialized weights; and the LEAP model (***Pereira et al., 2019***). Our Stacked DenseNet model achieves high accuracy using few training examples without the use the transfer learning. Using pretrained weights does slightly decrease overall prediction error for the DeepLabCut model (***Mathis et al., 2018***), but the effect size is relatively small.

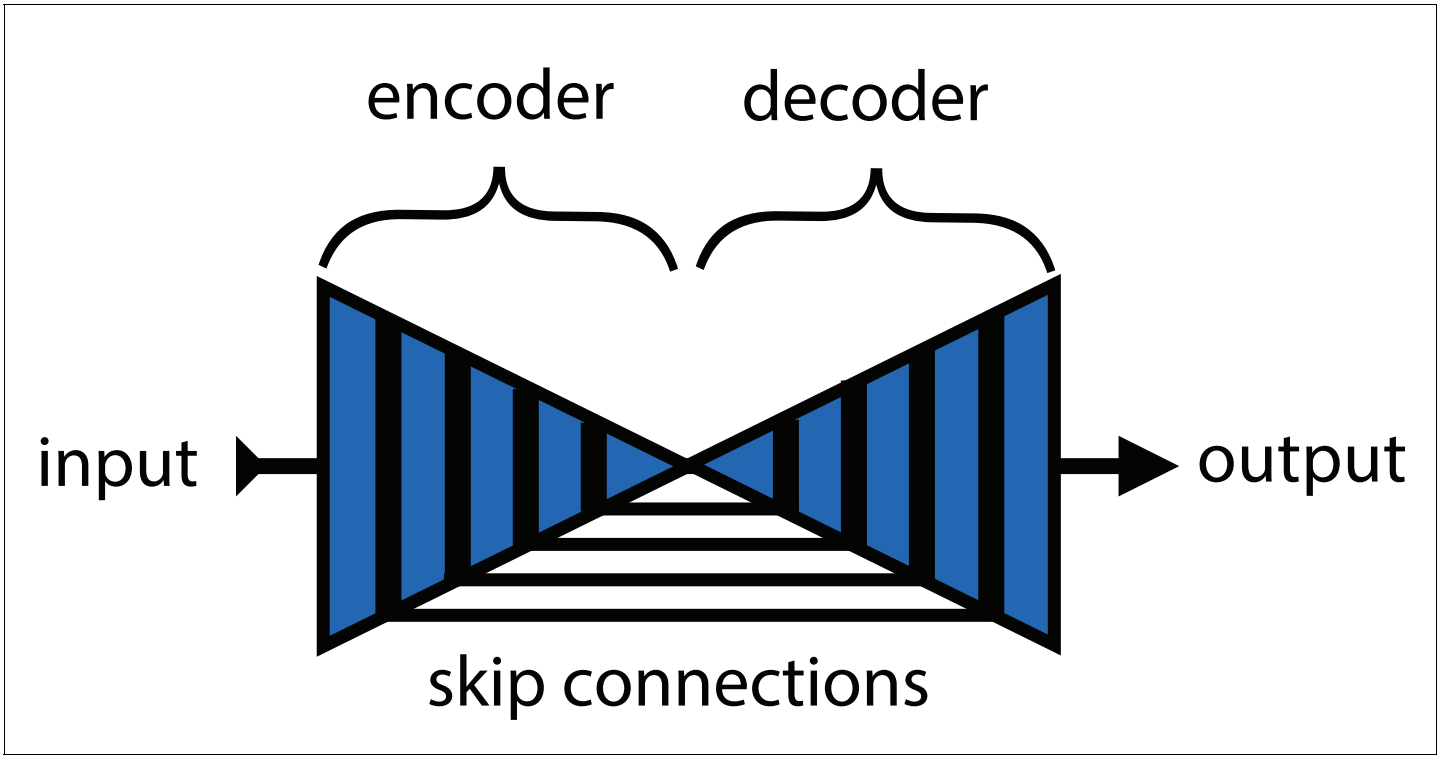


Appendix 4—figure 1. An illustration showing the progression of encoder-decoder architectures from the literature—ordered by performance from top to bottom (see Appendix 4: 'Encoder-decoder models' for further details). Most advances in performance have come from adding connections

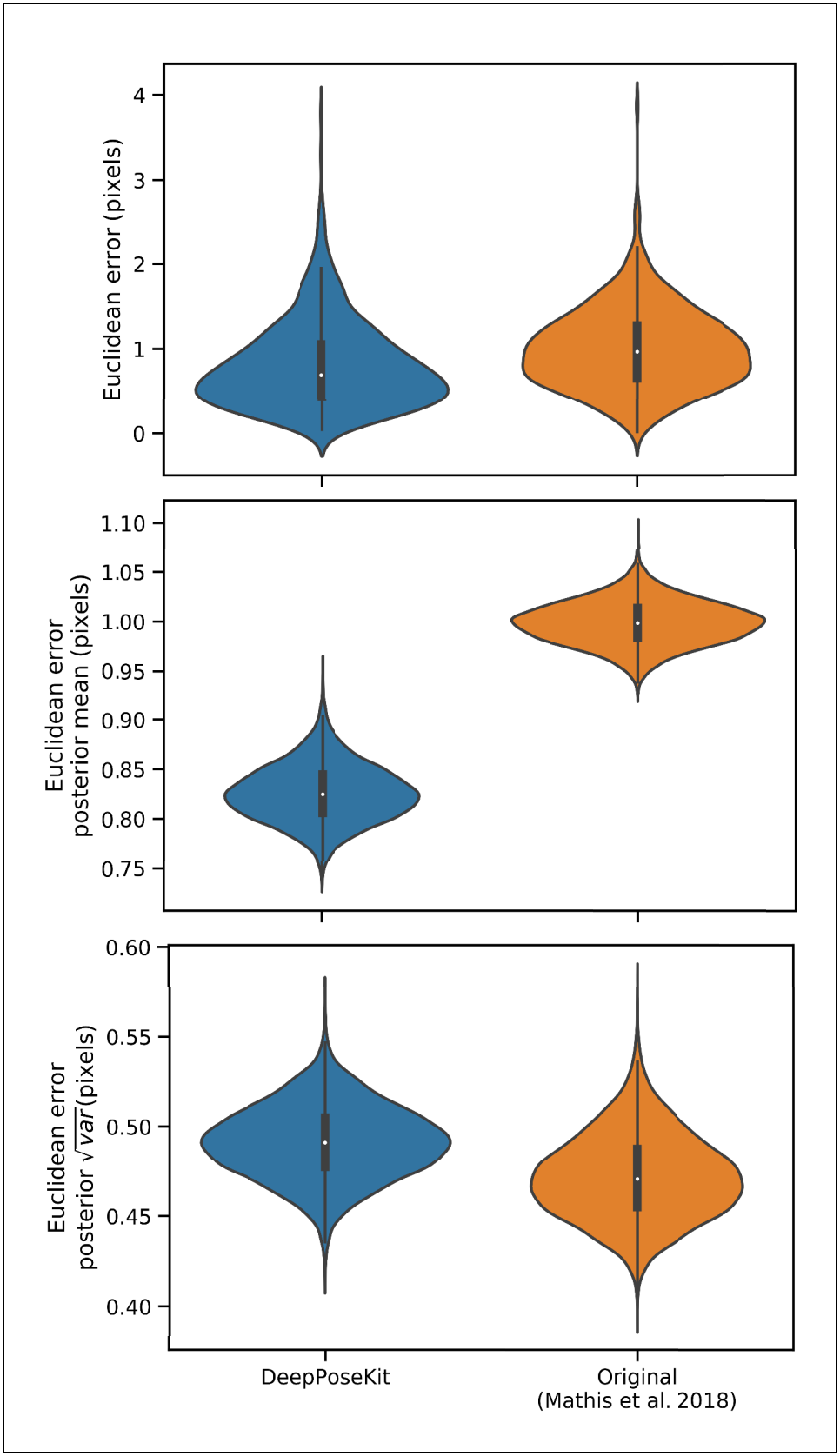
Appendix 4—figure 1 continued on next page

Appendix 4—figure 1 continued

between layers in the network, culminating in FC-DenseNet from ***Jégou et al. (2017)***. Lines in each illustration indicate connections between convolutional blocks with the thickness of the line indicating the magnitude of information flow between layers in the network. The size of each convolution block indicates the relative number of feature maps (width) and spatial scale (height). The callout for FC-DenseNet (***Jégou et al., 2017***; bottom-left) shows the elaborate set of skip connections within each densely-connected convolutional block as well as our additions of bottleneck and compression layers (described by ***Huang et al., 2017a***) to increase efficiency (Appendix 8).



Appendix 4—figure 2. An illustration of the basic encoder-decoder design. The encoder converts the input images into spatial features, and the decoder transforms spatial features to the desired output.

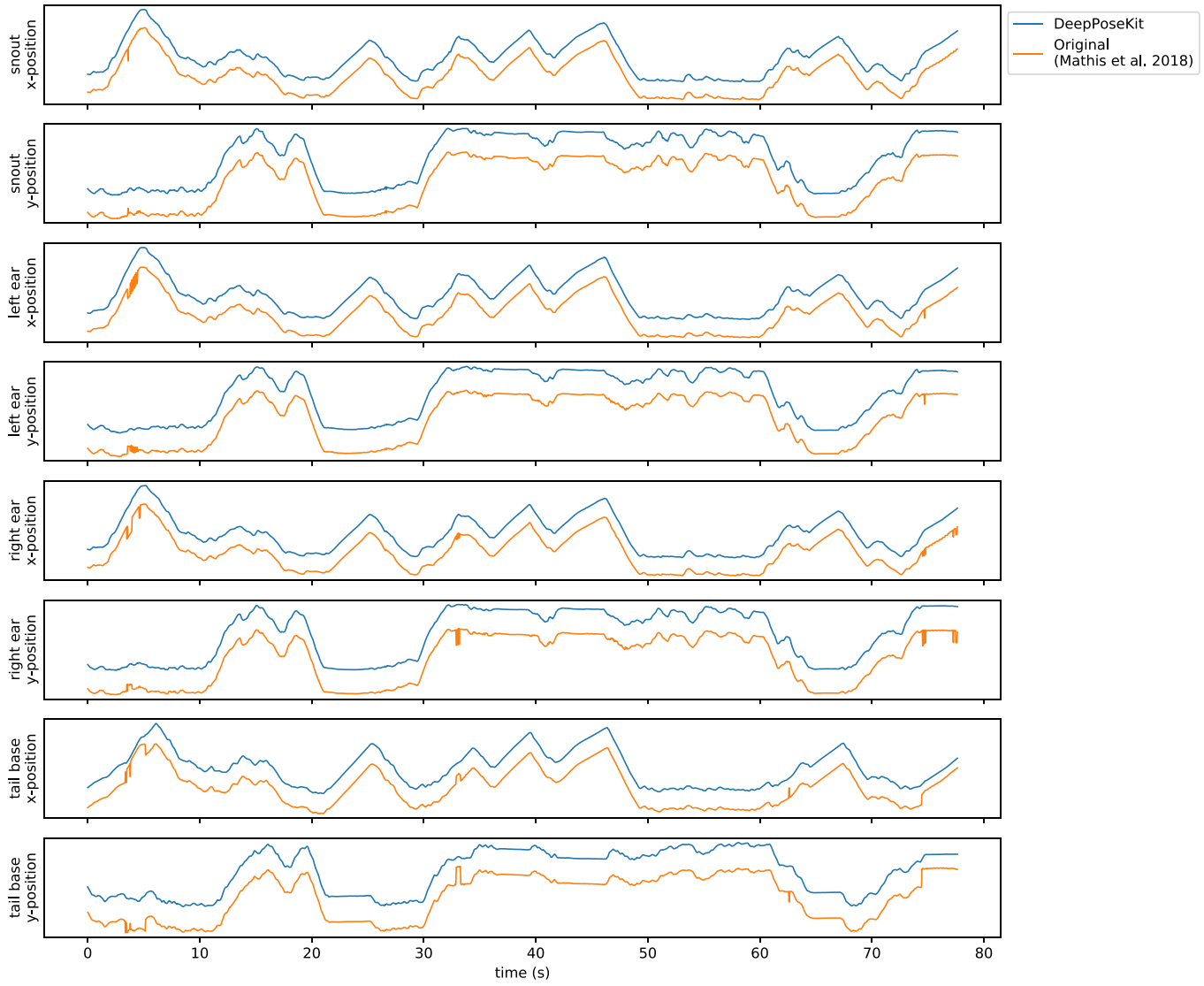


Appendix 8—figure 1. Prediction errors for the odor-trail mouse dataset from ***Mathis et al. (2018)*** using the original implementation of the DeepLabCut model (***Mathis et al., 2018***; ***Nath et al., 2019***) and our modified version of this model implemented in DeepPoseKit. Mean prediction

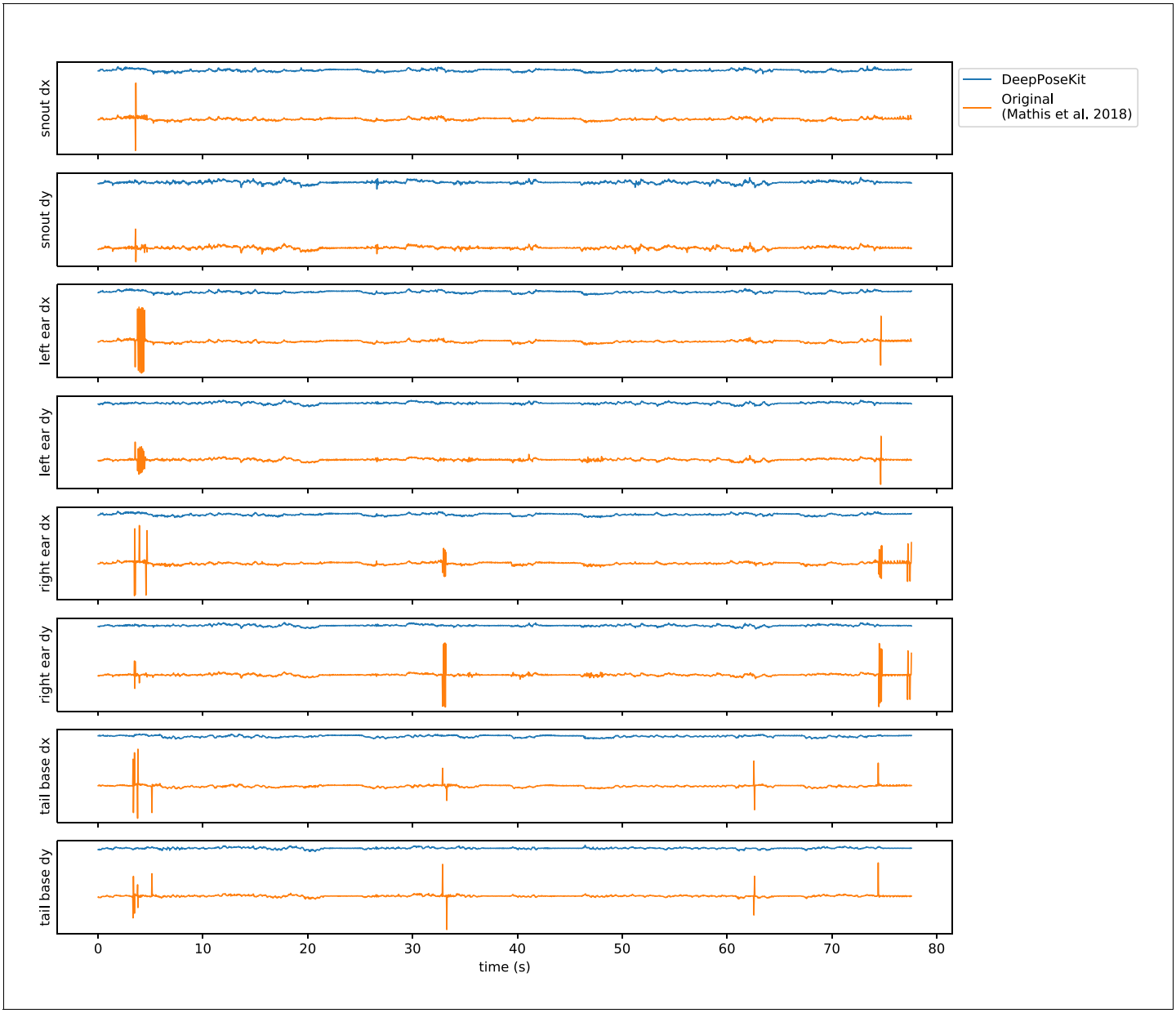
Appendix 8—figure 1 continued on next page

Appendix 8—figure 1 continued

error is slightly lower for the DeepPoseKit implementation, but there is no discernible difference in variance. These results indicate that the models achieve nearly identical prediction accuracy despite modification. We also provide qualitative comparisons of these results in ***Appendix 8—figure 1—figure supplement 1*** and ***2***, and ***Appendix 8—figure 1—video 1***.



Appendix 8—figure 1—figure supplement 1. Plots of the predicted output for ***Appendix 8—figure 1—video 1*** comparing our implementation of the DeepLabCut model (***Mathis et al., 2018***) in DeepPoseKit *vs.* the original implementation from ***Mathis et al. (2018)***, and ***Nath et al. (2019)***. Note the many fast jumps in position for the original verison from ***Mathis et al. (2018)***, which indicates prediction errors.



Appendix 8—figure 1—figure supplement 2. Plots of the temporal derivatives of the predicted output for ***Appendix 8—figure 1—video 1*** comparing our implementation of the DeepLabCut model (***Mathis et al., 2018***) in DeepPoseKit vs. the original implementation from ***Mathis et al. (2018)***, and ***Nath et al. (2019)***. Note the many fast jumps in position for the original verison from ***Mathis et al. (2018)***, which indicates prediction errors.