Article:

He, Y. orcid.org/0000-0003-3464-7526, Cooney, C.R. orcid.org/0000-0002-4872-9146, Maddock, S. et al. (1 more author) (2025) PhenoLearn: a user-friendly toolkit for image annotation and deep learning-based phenotyping for biological datasets. Journal of Evolutionary Biology. voaf058. ISSN 1010-061X

https://doi.org/10.1093/jeb/voaf058

PhenoLearn: A user-friendly Toolkit for Image Annotation and Deep Learning-Based Phenotyping for Biological Datasets

Authors: Yichen He[1,2]*, Christopher R. Cooney[1], Steve Maddock[3], Gavin H. Thomas[1,4]

[1] Ecology and Evolutionary Biology, School of Biosciences, University of Sheffield; Alfred Denny Building, Western Bank, Sheffield S10 2TN, UK.

[2] Department of Life Sciences, Natural History Museum; Cromwell Rd, South Kensington, London SW7 5BD, UK

[3] Department of Computer Science, University of Sheffield; Regent Court, 211 Portobello, Sheffield S1 4DP, UK.

[4] Bird Group, Department of Life Sciences, Natural History Museum; Akeman Street, Tring, HP23 6AP, UK.

*Corresponding authors. Email: csyichenhe@gmail.com

Acknowledgements. We thank Thomas Guillerme, Kathryn Harris and Eleftherios Ioannou for testing this toolkit.

Funding. This work was funded by a Leverhulme Early Career Fellowship (ECF-2018-101) and Natural Environment Research Council Independent Research Fellowship (NE/T01105X/1) to C.R.C, and a European Research Council grant (615709, Project 'ToLERates') and Royal Society University Research Fellowship (UF120016, URF\R\180006) to G.H.T.

Authors contributions. All authors designed the tool; Y.H. developed the tool; C.R.C. and G.H.T. tested the tool. Y.H. wrote the manuscript with input from all authors.

Competing interests. All authors have no competing interests.

Data and materials availability. All code, datasets, and binaries used in this study are publicly archived and available:

- **Source code:** The PhenoLearn source code is available on GitHub for continued development: https://github.com/echanhe/phenolearn. A snapshot of the code corresponding to the version used in this paper has been archived on Zenodo and assigned a DOI: https://zenodo.org/records/15350513.
- Example datasets: The bird and Littorina test datasets used for evaluation are available on Zenodo: https://zenodo.org/records/8152784.
- Binary executable: The compiled Windows binary of the PhenoLabel annotation tool is also archived on Zenodo: https://zenodo.org/records/10909841.

Abstract

The digitisation of natural history specimens has unlocked opportunities for large-scale phenotypic trait analysis. In recent years, deep learning has shown significant results in accurately predicting annotations on 2D specimen photographs. However, it can be challenging for biologists without extensive related expertise to easily use deep learning. Here, we introduce PhenoLearn, a toolkit developed for biologists to generate annotations on 2D specimen images using deep learning. PhenoLearn integrates graphical user interfaces (GUIs) within its two main modules, PhenoLabel for image annotation and PhenoTrain for model training and prediction. GUIs increase accessibility and reduce the need for computational expertise, allowing biologists to intuitively go through a workflow of labelling training sets, using deep learning, and reviewing predictions in the same tool. We demonstrate PhenoLearn's capabilities through a case study involving the segmentation of plumage areas on bird images, showcasing prediction accuracy and the running time with and without GPU, highlighting its potential to generate annotations with minimal computational cost and time. The toolkit's modular design and flexibility ensure adaptability, allowing for integration with other tools amidst rapidly evolving deep learning approaches. PhenoLearn bridges the gap between specimen digitisation and downstream analysis, providing biologists with broader access to deep learning. The source code, installation guides, tutorials with screenshots, and a small demo dataset for PhenoLearn can be found at https://github.com/echanhe/phenolearn.

Keywords: Deep Learning, Phenotyping, Image Annotation, Phenotypic Trait, Toolkit with User Interface.

Introduction

The process of measuring phenotypic traits on 2D digitised specimen images is increasingly used to phenotype specimens for a range of tasks. Through the use of annotations such as points (Chang & Alfaro, 2016; Zelditch et al., 2004) and segmentations (Cooney et al., 2022; Y. He et al., 2022), researchers can extract and analyse a variety of morphological measurements from specimens to provide insights into evolutionary and ecological questions. Digitisation allows rapid and non-invasive measurements of natural history collections and mobilises specimens for further analyses, helping to unlock their full potential. Techniques such as tray scanning (Blagoderov et al., 2012) have significantly accelerated the digitisation of entomological collections, by leveraging robotic automation to automatically capture 2D images of specimens directly from museum trays. In addition, many computational tools for analysing phenotypes like shape (Adams & Otárola-Castillo, 2013) and colouration (Maia et al., 2019) have been developed, expanding the breadth of tools available to analyse phenotypic traits. However, manually pre-processing images (e.g., placing annotations) is time-consuming, especially with large datasets such as hundreds of thousands of observations (Cooney et al., 2022). To prevent manual annotation from becoming a bottleneck for mobilising large digital datasets, efficient high-throughput data extraction tools are essential.

Classic computer vision algorithms like thresholding, connected components and region growing have been used for extracting phenotypic information from images, representing a significant increase in measurement speed compared to manual methods (Lürig, 2022; Pennekamp & Schtickzelle, 2013). Deep learning-based methods have recently become state-of-the-art for various computer vision tasks, including object segmentation in images with complex backgrounds. In particular, deep learning applications for measuring digitised specimens have demonstrated success with different types of annotations, including points (Mathis et al., 2018; Porto & Voje, 2020), bounding boxes (John et al., 2024; Shedrawi et al., 2024) and segmentations (Y. He et al., 2022; Schwartz & Alfaro, 2021). These methods yield high-throughput pipelines and accurate results, illustrating the potential for expanding deep learning to other biological datasets. However, several barriers remain preventing the widespread application of deep learning in ecology and evolutionary biology.

A significant barrier to the wider adoption of deep learning is the generally high level of technical skill required for implementation. This issue is often compounded by the lack of intuitive platforms that allow non-specialists to use deep learning for phenotyping. Recent phenotyping toolkits, such as DeepLabCut (Mathis et al., 2018) and Argos (Ray & Stopfer, 2022), have focused on improving accessibility through graphical user interfaces (GUIs). The integration of deep learning models with GUIs can greatly increase accessibility, allowing researchers with limited technical knowledge to utilise these advanced techniques. Furthermore, the development of fully-integrated toolkits for performing a complete workflow, including labelling training sets, training models and reviewing predictions, can significantly improve the accessibility and efficiency for biologists to apply deep learning in biological research. Such a toolkit, tailored for extracting traits from 2D specimen photographs for ecological and evolutionary studies, would serve as a much-needed bridge between digitisation and downstream biological analysis.

Here, we introduce PhenoLearn, a user-friendly toolkit for generating annotations using deep learning. PhenoLearn comprises two main modules, PhenoLabel and PhenoTrain, covering three main functions (**Figure 1**). PhenoLabel implements both image labelling and reviewing, whereas PhenoTrain implements the functions for deep learning. As an open-source tool with GUIs, PhenoLearn aims to minimise the computational expertise required to generate point or segmentation predictions using deep learning for 2D biological image datasets. While PhenoLearn is designed to facilitate the entire annotation generation workflow, its modular design allows users to use individual functions for desired tasks. For instance, PhenoLabel can be used to review predictions from other methods. Likewise, labels generated elsewhere can be used to train models implemented in PhenoTrain. The PhenoLearn pipeline has already been successfully used to generate annotations in several large-scale research projects (Cooney et al., 2022; Y. He et al., 2022, 2023). In this paper, we provide a detailed explanation, a user guideline, and an example of using PhenoLearn.

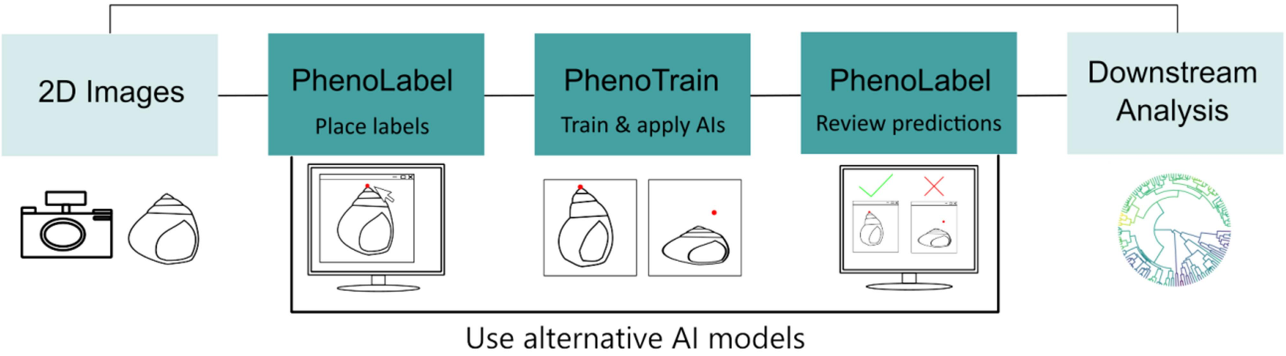


Figure 1. Workflow overview of using PhenoLearn to generate annotations for biological datasets. Green boxes indicate steps involving PhenoLearn modules (PhenoLabel and PhenoTrain), which offer a connection between digitisation (2D imaging) and downstream biological analysis.

Installation

PhenoLearn was developed using Python 3, with the following libraries and their versions tested during its development:

- Python: 3.10
- PyQt: 5.15.9
- NumPy: 1.25.1
- pandas: 2.0.3
- opencv-python: 4.8.0.74
- PyTorch: 2.0.1
- TensorBoard: 2.13.0

For deep learning, PhenoLearn is optimised to utilise NVIDIA graphics processing units (GPUs) through CUDA (https://developer.nvidia.com/cuda-toolkit). While it is possible to train models using the CPU on systems without CUDA-supported GPUs, this will generally lead to slower running time. We recommend using a GPU with at least 8GB of video memory for faster running time.

PhenoLearn's two main modules, PhenoLabel and PhenoTrain have their own GUIs. PhenoLabel implements the labelling and reviewing functions and can be accessed by running **phenolabel.py**. PhenoTrain handles deep learning training and prediction and is accessed by running **phenotrain.py**. It was tested on Windows 10, macOS 13.6, and Ubuntu 22.04.3 LTS.

The source code, installation guides, tutorials with screenshots, and a small demo dataset for PhenoLearn can be found at https://github.com/echanhe/phenolearn. Datasets used in the example section are available at https://zenodo.org/records/8152784. For Windows users, a binary version of PhenoLabel (e.g., a .exe file) is available at https://zenodo.org/records/10909841. Detailed file introductions can be found in the supplementary material.

Design and Implementation

Labelling

This section outlines using PhenoLabel for labelling, including creating a project, placing points/segmentations, and managing progress. To start, select 'Open Dir' in the File menu (**Figure 2**a) to open a folder of images for labelling. PhenoLabel uses the imread function from opencv-python (Bradski, 2000), which supports common formats including jpg, png and tiff. PhenoLabel lists all images in the File panel (**Figure 2**d) and displays the selected image in the Main panel (**Figure 2**e). Users can zoom the image and view the cursor coordinate and RGB values in the status bar (**Figure 2**g).

Users can place points or segmentations in the Main panel. For points, click the 'Point' button on the Toolbar (**Figure 2**b) and left-click the image. Points can be named via a dialogue box, either inputting a new name or selecting from a dropdown menu. Existing points can be modified or deleted in the Annotation panel (**Figure 2**f). PhenoLearn records vertical (y) coordinates from the top of the image downward. This is the standard convention used in many Python-based image processing libraries, such as opencv-python. In contrast, tools like tpsDig (Rohlf, 2006) and many R-based image analysis tools typically record coordinates from the bottom up. Users working with tpsDig datasets should be aware of this difference.

For segmenting, click the 'Segmentation' button on the Toolbar, and a Segmentation Toolbar (**Figure 2**c) appears. Segmentation classes must be named using the 'Add' button in the Segmentation tab (**Figure 2**f). Then by activating the 'Draw' button in the Segmentation Toolbar and holding the left mouse button, users can use a paintbrush to draw the region of interest (ROI). Each segmented ROI is automatically assigned a distinct colour, allowing users to easily differentiate between them. Segmented areas can be removed with the same operation with the 'Erase' button activated. To efficiently segment large areas, users can outline a region and then use the 'Auto Fill' function to fill the area within the outline. Four paintbrush sizes are available: S, M, L, and XL.

PhenoLabel's 'Fast Labelling' function automates annotation naming for cases that use consistent annotation names, eliminating repeated manual naming. This feature automatically creates annotation names for subsequent images using the annotation names from the current image. To ensure a newly placed point matches the preset point names, they need to be placed in the same order as the names displayed at the bottom of the Annotation Panel.

'Save' and 'Save As' in the File menu allow users to save their work in JSON format, which includes details on images and annotations. 'Open Labelling Progress' allows users to continue or review their labelling progress. Annotations can be exported to PhenoTrain in CSV or binary masks (for single-class segmentations). Two types of CSV exports are available: a point CSV file and a segmentation CSV file. Refer to Table 1 for the detailed structure of the JSON and CSV files.

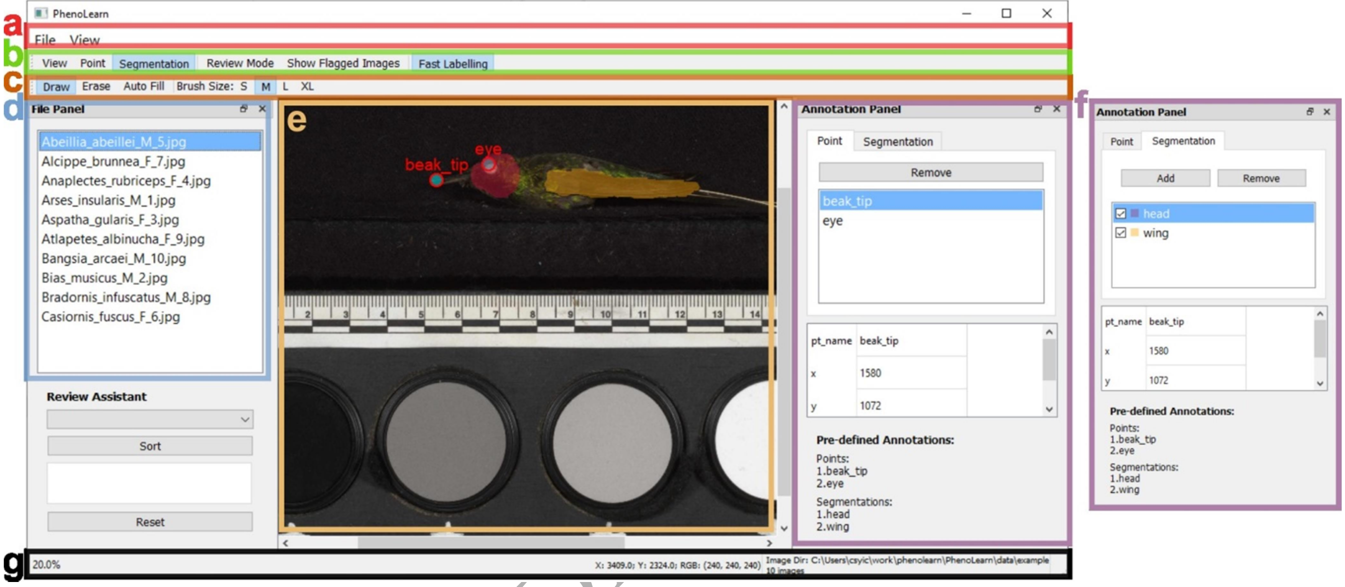


Figure 2. The PhenoLabel GUI. (a) Menu bar: Provides functions for saving projects and loading files, (b) Toolbar: Tools for image annotation manipulation, (c) Segmentation Toolbar: Tools specifically designed for segmentation tasks, (d) File panel: Displays the loaded images and allows users to switch between images, (e) Main panel: The central workspace for image annotation, (f) Left: Annotation panel, Point tab for displaying details of point-based annotations; Right: Annotation panel, Segmentation tab for displaying details of segmentation-based annotations, (g) Status bar: Displays status such as image name, zoom level, and cursor position.

Table 1. File structures used in PhenoLearn.

File	Description
Labelling progress file	A JSON file **From:** Created by the save function in PhenoLabel. **Usage:** Can be used to load the progress into PhenoLabel

Structure:
The file is a list of dictionaries.

- “file_name” stores the image name.
- “points” stores a list of dictionaries.
    - “name” stores the point name
    - “x” stores the x coordinate
    - “y” stores the y coordinate
    - “absence” stores if the point is missing
- “segmentations” stores a dictionary.
    - Dictionary keys are the names of the segmentations and dictionary values are the segmentations.

A segmentation is stored as a four-level nested list, which follows the format of segmentation contours extracted by OpenCV (Bradski, 2000). The format is:

- The first level corresponds to the segmentation itself.
- The second level is the contour level, where one segmentation may include one or more contours.
- The third and fourth levels pertain to the point level, with each contour having multiple points.

The example below shows a segmentation consisting of two contours. Contour 1 contains 'n' points, and Contour 2 contains 'm' points. Here <x_12> represents the x-coordinate of the second point in Contour 1.

Example:

```
[ {
"file_name": "Abeillia_abeillei_M_5.jpg",

"points":
  [
    {"name": "beak", "x": 1580, "y": 1072},
    {"name": "eye", "x": 1876, "y": 984}
  ],

"segmentations":
  {"head":
        [
          [
            [ [ <x_11>, <y_11> ] ],
            [ [ <x_12>, <y_12> ] ],  …
            [ [ <x_1n>, <y_1n> ] ]
          ] ,
```

```
              [
                [ [ <x_21>, <y_21> ] ],
                [ [ <x_22>, <y_22> ] ],  …
                [ [ <x_2m>, <y_2m> ] ]
              ]
            ]
      } ]
```

Output Point CSV file

A CSV file
From: Exported by PhenoLabel or generated by PhenoTrain.
Usage: Can be imported into PhenoTrain as for training.

Structure:
The “file” column stores the image names.
A “<point name>_x” column stores the x coordinate for a point.
A “<point name>_y” column stores the y coordinate for a point.
A value of -1 or an empty cell indicates the point is missing.

Example:

file	beak_x	beak_y	eye_x	eye_y
Abeillia_abeillei_M_5.jpg	1580	1072	1876	984

Output segmentation CSV file

A CSV file
From: Exported by PhenoLabel or generated by PhenoTrain.
Usage: Can be imported into PhenoTrain for training.

Structure:
The “file” column stores the image names.
The remaining columns store the segmentations.

A segmentation is stored as a four-level nested list.
The details and examples can be found in the “Labelling progress file” row. Here, the example only shows a four-level nested list placeholder for better readability.

Example:

file	head
Abeillia_abeillei_M_5.jpg	[[[[ ]]]]

Output binary mask

A black and white image
From: Exported by PhenoLabel or generated by PhenoTrain.
Usage: Can be imported into PhenoTrain for training.

<table>
<tr><td></td><td>A grayscale image is saved under the same name as its input image, with background areas in black and segmentation areas in white. To prevent having the output masks replace the input images, ensure the input directory is not used as the output directory.</td></tr>
<tr><td>Property file</td><td>A CSV file.
Usage: Import specific specimen properties into PhenoLabel to filter or sort images, allowing users to prioritise error-prone images first.

Structure:
The “file” column stores the image names.
Other columns store the properties.
Categorical properties are stored as text strings.
Numerical properties are stored as numbers.
Example:<table><tr><th>file</th><th>id</th><th>sex</th></tr><tr><td>Abeillia_abeillei_M_5.jpg</td><td>5</td><td>M</td></tr></table></td></tr>
</table>

Deep Learning

PhenoTrain allows users to train models and make predictions. This section demonstrates how to set up model training and prediction in PhenoTrain.

Model Training

Before training, eleven settings are required via the Train tab of PhenoTrain (**Figure 3**a). Some settings have default values derived from previous studies (Chen et al., 2017; K. He et al., 2017; Y. He et al., 2022, 2023) and the PyTorch documentation (Paszke et al., 2019). These defaults provide a solid starting point for various applications:

(1) *Model type*. Mask R-CNN (K. He et al., 2017) for point and DeepLabv3 (Chen et al., 2017) for segmentation. Despite the availability of numerous new deep learning architectures, we use Mask R-CNN and DeepLabv3 for their robust nature and adaptability to various tasks. Being well-established models, there are many tutorials available online that facilitate their implementation for users who want to understand the detailed information.

(2) *Annotation Input format*. The default option is CSV. For single-class segmentations, 'Mask' option is also available for using binary masks as inputs. Please refer to Table 1 for the details of the binary mask.

(3) *Annotation file*. The CSV annotation file from PhenoLabel (only applicable when ‘CSV is selected for Setting 2).

(4) *Mask folder*. The folder of the binary masks (only applicable when ‘Mask’ is selected for Setting 2).

(5) *Image folder.* The folder of training images.

(6) *Image resize percentage*. Ranges from 1-100%, keeps aspect ratio, using nearest neighbour interpolation.

(7) *Validation set percentage*. The percentage of validation images used for evaluating the model per epoch. A common split is 80/20 for training/validating.

(8) *Batch size*. The number of images processed in one training iteration. The default is 1. A smaller batch size saves memory but may lead to less stable optimisation. Conversely, a larger batch size may provide better optimisation, but it uses more memory. Users need to test a set of batch sizes to find the optimal value.

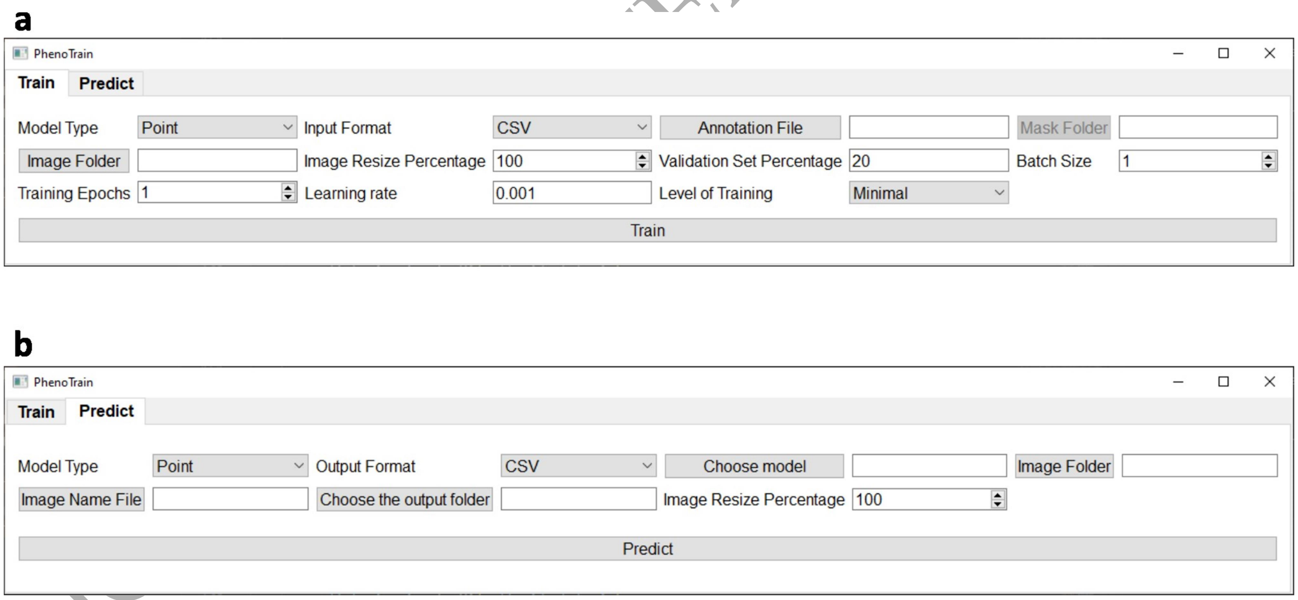
(9) *Training epochs*. The number of times the entire training set passes through the model. Training for more epochs may lead to better model performance. The default training epoch is set to 1. Users can estimate the training time by training for one epoch.

(10) *Learning rate*. Controls the step size during the optimisation phase of training. The default learning rate for PhenoTrain is 0.001. A too-large learning rate may result in overly large steps, causing the model to miss the optimum. A too-small learning rate might lead to a very slow convergence towards the optimum.

(11) *Level of training*. Controls the proportion of the model that is trained. The options are Minimal, Intermediate and Full. "Minimal" trains only the final layers, "Intermediate" trains half of the model layers, and "Full" trains the entire model.

(12) *CPU/GPU*. Select whether to use the CPU or GPU for training. If GPU is selected but no GPU is available on the device, CPU will be used.

When the training is completed, a .pth file is saved in the ‘saved_model’ folder.



a

Model Type: Point
Input Format: CSV
Image Resize Percentage: 100
Validation Set Percentage: 20
Batch Size: 1
Training Epochs: 1
Learning rate: 0.001
Level of Training: Minimal

b

Model Type: Point
Output Format: CSV
Image Resize Percentage: 100

Figure 3. The PhenoTrain GUI. The interface has two tabs, (a) the Train tab and (b) the Predict tab. Settings for training and predicting can be specified in each tab.

The training level setting utilises transfer learning (Tan et al., 2018), focusing on training with a pre-trained model. Transfer learning diverges from the approach of using randomly initialised model weights, which generates poor initial predictions and can take a longer training period. Instead, it leverages a pre-trained model, which effectively gives the model prior knowledge gained from previous tasks. This approach can train on parts of a model and achieve satisfactory results, saving both time and computational resources. Both DeepLabV3 and Mask-R-CNN were pre-trained on the COCO dataset (Lin et al., 2014), which is a large-scale image dataset for computer vision tasks such as segmentation.

PhenoTrain integrates with TensorBoard (Martín Abadi et al., 2015) to visualise the training progress. Logs are saved in the 'runs' folder. To view logs in TensorBoard, run this command: **`tensorboard --logdir==runs`** in python. Upon execution, it can be viewed in a web browser at http://localhost:6006/. Users can view and compare across different training runs.

TensorBoard saves training and validation loss, along with evaluation metrics. Training loss indicates the model's learning efficiency, while validation loss evaluates performance on the validation set. Point accuracy is assessed using the pixel distance (Euclidean distances between two points on an image). The Dice Score is used to evaluate segmentations, based on the overlap between predicted and manual segmentations. The Dice Score ranges from 0 (lowest) to 1 (highest). Average and class-specific metrics for points or segmentations are stored.

Generating Predictions

Once a well-trained model is saved, users can generate predictions in the Predict tab (**Figure 3**b) by configuring the following seven settings:

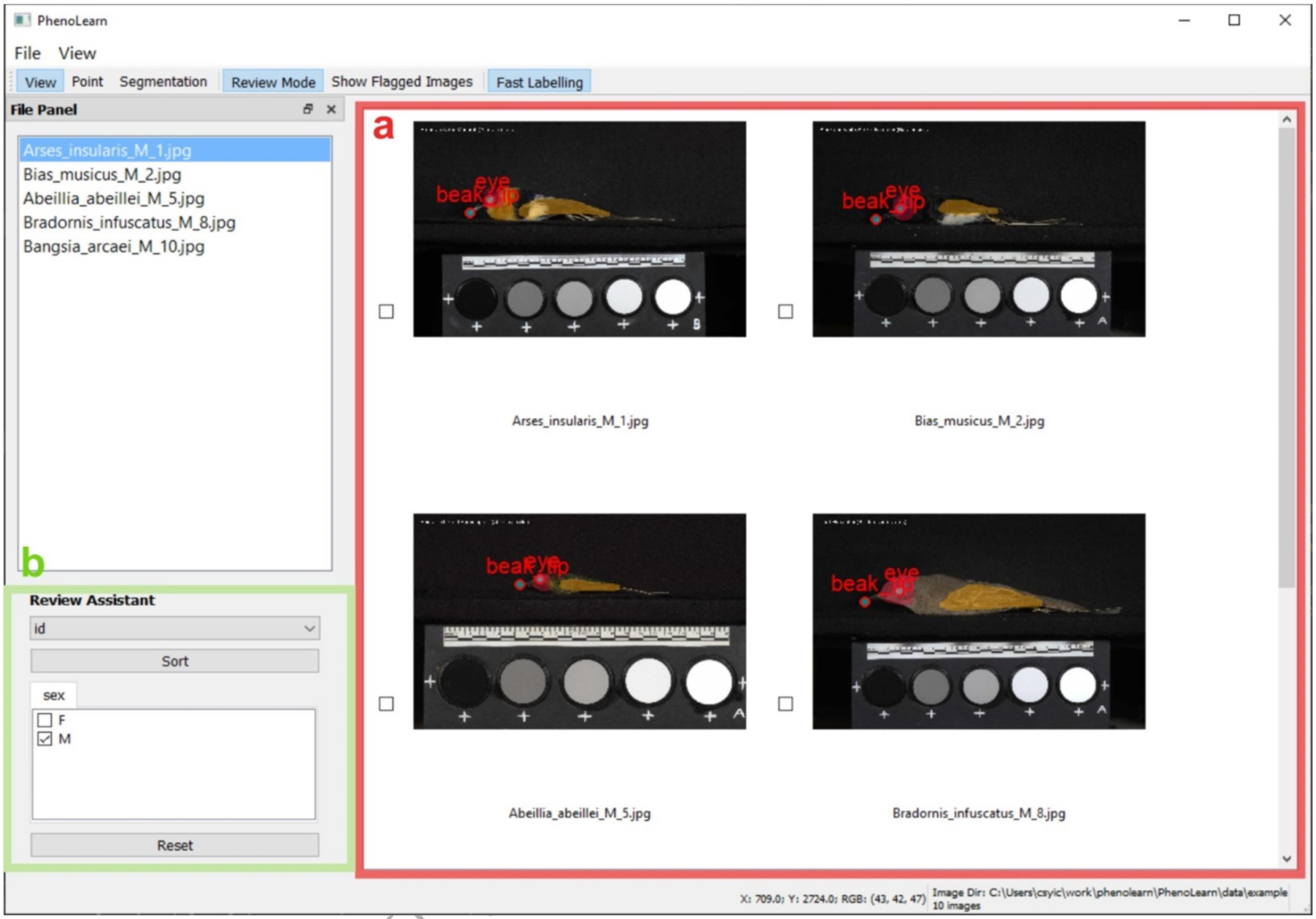
(1) *Model type.* Point or Segmentation.
(2) *Output format*. Options are CSV file or mask images (for single-class segmentations only).
(3) *Choose model.* .pth file saved from training.
(4) *Image folder*. The folder of images for prediction.
(5) *Image name file*. CSV file with one column named 'file' for image names. PhenoLabel can export an Image name file when no annotations are presented for the images.
(6) *Choose the output folder.* A folder for the prediction file.
(7) *Image resize percentage.* Ranges from 1-100% and should be consistent with the percentage used in training.
(8) CPU/GPU. Select whether to use the CPU or GPU for predicting. If GPU is selected but no GPU is available on the device, CPU will be used.

PhenoTrain provides real-time updates during both training and prediction phases, including a progress bar and elapsed time display.

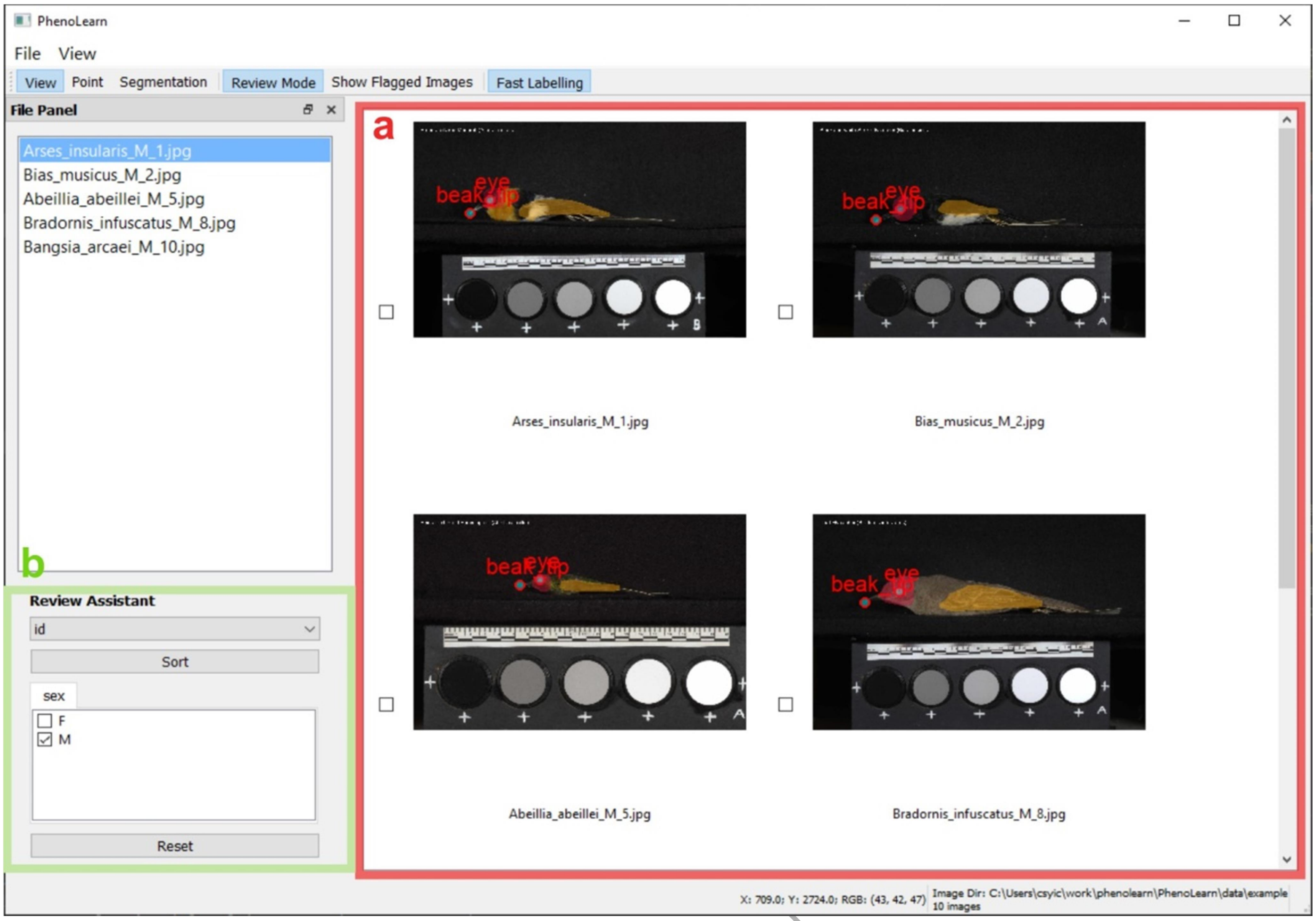
Reviewing predictions

Deep learning predictions are not perfectly accurate, and reviewing predictions is often necessary to confirm and/or improve accuracy for biological applications. To facilitate this, we have incorporated two features within PhenoLabel: (1) Review Mode and (2) Review Assistant to improve reviewing efficiency.

Users can open an image folder and import predictions (e.g., outputs from PhenoTrain) into PhenoLabel, and subsequently review and improve these predictions. By activating the Review Mode in the Toolbar, PhenoLabel displays multiple image thumbnails with annotations (**Figure 4** **Figure 4**a). In this mode, users can quickly browse through images and flag any with incorrect predictions by ticking adjacent checkboxes. After checking through thumbnails, click 'Show Flagged Images' button to show only the flagged images for a more focused review. Additionally, it is possible to export the predicted annotations for input into other outlier detection methods and to create flagged images.



The Review Assistant improves review efficiency by leveraging specimen metadata. By prioritising images with specific properties (e.g. a problematic species), users can optimise accuracy and time efficiency. The Review Assistant facilitates this by offering options to sort or filter images based on properties (**Figure 4**



Review Assistant: id

sex
☐ F
☑ M

X: 709.0; Y: 2724.0; RGB: (43, 42, 47) Image Dir: C:\Users\csyic\work\phenolearn\PhenoLearn\data\example 10 images

Figure 4b), which can be imported from a property file (**Table** 1). It can sort images by numerical properties (e.g. specimen length) and filter images by categorical properties (e.g. taxa). The 'Reset' button clears all filters and sorting.

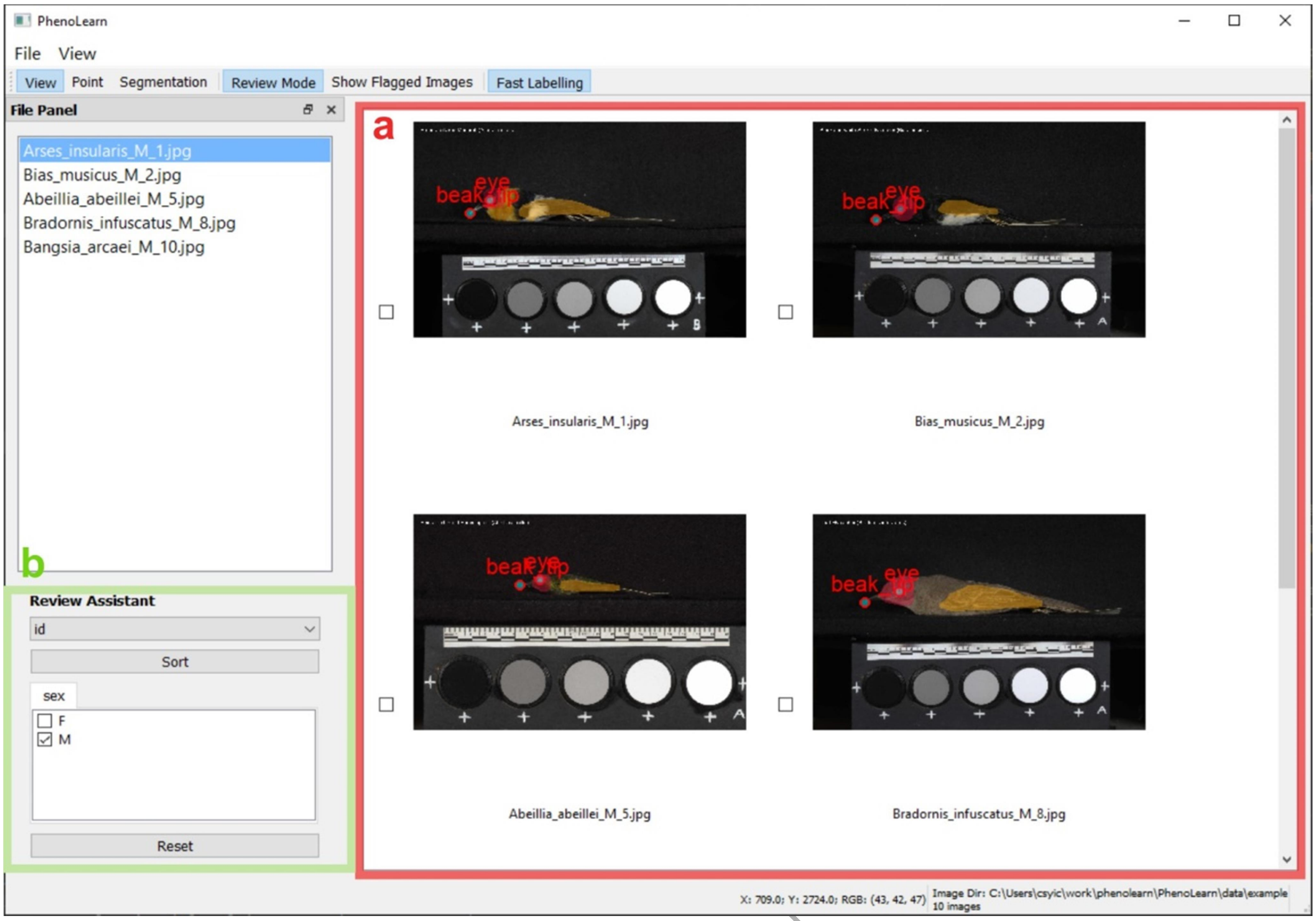


Figure 4. The PhenoLabel GUI with Review Mode activated. (a) The Review panel, which replaces the Main panel, displays image thumbnails with annotations. (b) The Review Assistant. In this example, it is used to select male specimens and sort images by ID.

Examples

The examples described below were executed on a Windows 10 system featuring an Intel(R) Core(TM) i7-11800H CPU, 16 GB of RAM, and an RTX 3080 GPU with 16 GB of video memory (VRAM). For memory usage results, the highest memory allocation observed in Task Manager was recorded for CPU usage, while GPU memory usage was from the output of the nvidia-smi command.

Segmenting with PhenoLearn

We tested PhenoLearn on a dataset of 220 bird images (4948 x 3280 pixels) to segment the whole plumage area. We used 120 images for training and the remaining 100 images for prediction. The 120 training images were annotated in PhenoLabel for training. The DeepLabv3 model was trained for five epochs, with a 20% validation set, batch size of two, 0.001 learning rate, minimal training level, and an input resolution of 494 x 328 pixels (10% downsampling).

The training process was faster with GPU, taking 3 minutes, compared to 13 minutes without it (CPU only). Predictions were generated in under a minute with GPU and 4 minutes without it. Examples of the predictions can be found in Figure 5. One of the authors (Y.H.) spent five minutes reviewing 100 images. An additional four minutes were used to correct predictions for these 18 images. In addition, we tested the training time, GPU usage and performance for using various configurations of GPU and CPU with different training levels to users with a comprehensive reference. The results are summarised in Table 2.

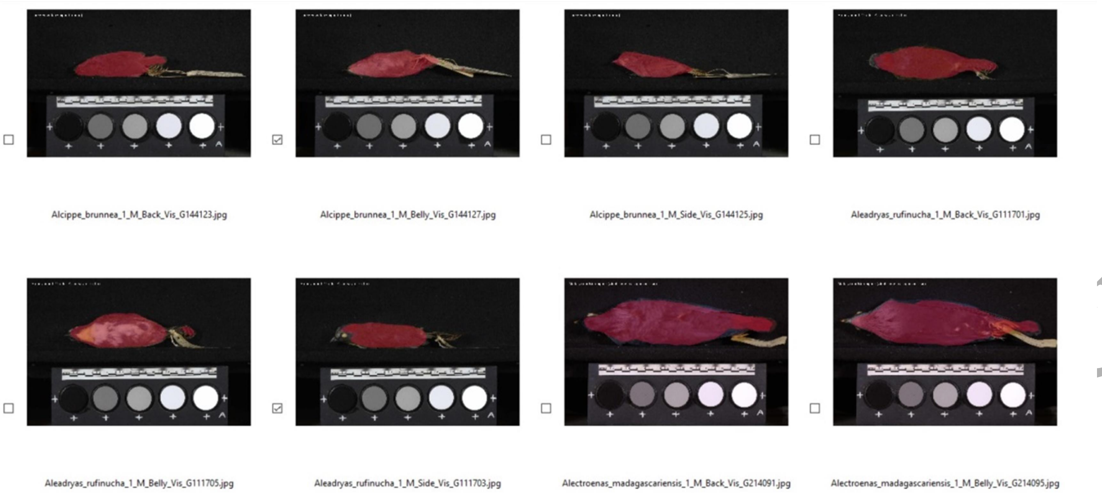


Figure 5. Examples of the segmentation predictions in the review mode.

Table 2. Training time, memory usage (RAM for CPU and VRAM for GPU), and performance across different hardware configurations and training levels on the segmentation test dataset.

Training Level	***Hardware***	***Training Time (Minutes)***	***Memory Usage (GB)***	***Average Dice Score***
Minimal	GPU	2	1	0.90
	CPU	13	1.2	
Intermediate	GPU	3	3.5	0.94
	CPU	26	4.1	
Full	GPU	3	3.9	0.93
	CPU	30	4.5	

Placing points with PhenoLearn

We evaluated PhenoLearn on a dataset of 220 *Littorina* images, each measuring 2592 x 1944 pixels, with four points annotated on each image according to a 15-landmark scheme derived from Ravinet et al. (2016). For this study, 120 images were used for training, while the remaining 100 served for prediction. Annotations for the training images were performed using PhenoLabel.

We trained a Mask R-CNN model over five epochs, using a validation set comprising 20% of the data, a batch size of two, a learning rate of 0.001, and an input resolution reduced to 518 x 388 pixels (20% downsampling). We conducted experiments using both GPU and CPU across various training levels. The best performance was an average pixel distance of 21. Details on GPU usage and the performance of different runs can be found in Table 3.

Table 3. Training time, memory usage (RAM for CPU and VRAM for GPU), and performance across different hardware configurations and training levels on the point test dataset.

Training Level	***Hardware***	***Training Time (Minutes)***	***Memory Usage (GB)***	***Average Pixel Distance***
Minimal	GPU	2	2.7	138
	CPU	15	1.5	
Intermediate	GPU	2	3	126
	CPU	25	1.9	
Full	GPU	3	4.5	37
	CPU	29	3.5	

Examples of the predictions made using PhenoLearn are illustrated in Figure 6. One of the authors (Y.H.) spent five minutes reviewing 100 *Littorina* images, during which 19 images with inaccurately placed points were flagged. An additional four minutes were spent to correct these predictions.

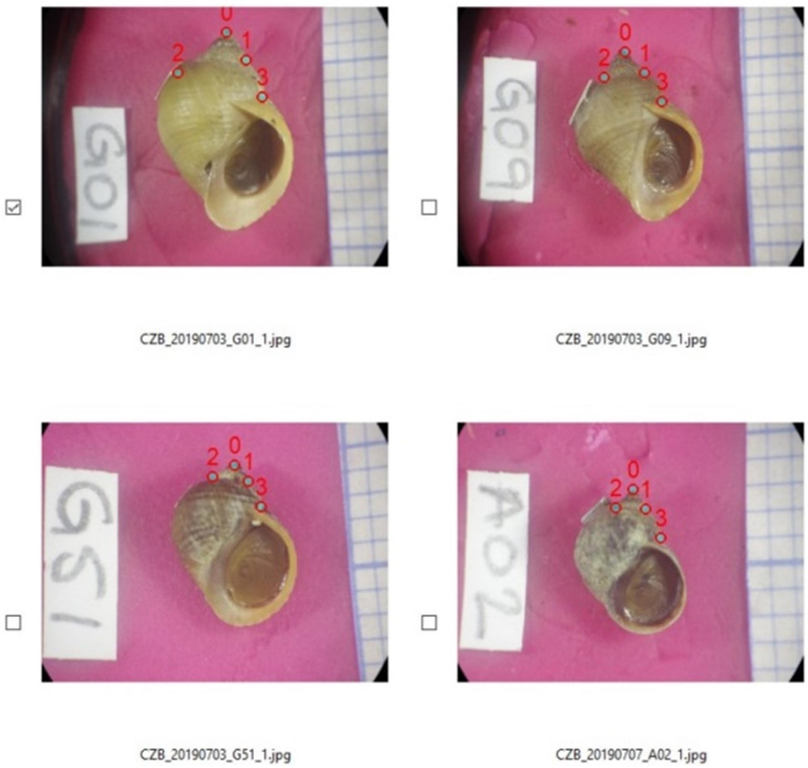


Figure 6. Examples of the point predictions in the review mode.

The performance of PhenoTrain can vary with different datasets and training settings. As shown in the results, training from scratch is not guaranteed to outperform fine-tuning pre-trained models (see Table 2). The pre-trained models used in PhenoLearn are based on the ImageNet dataset (Deng et al., 2009), which provides a large and diverse set of features as a strong starting point. Pre-trained models are also less prone to overfitting and more capable of generalising to new datasets (Huh et al., 2016; Yosinski et al., 2014). This advantage makes fine-tuning a pre-trained network a reliable choice in many scenarios. However, the relative performance of these approaches can only be determined through testing. Based on our observations, we recommend starting with fine-tuning for most use cases and minimum computational cost.

Another important point is that the randomness inherent in the training process, such as random weight initialisation and data shuffling during batch creation, can lead to variability in results. Even with identical configurations and training data, different runs may yield slightly different outcomes. This variability should be considered when interpreting results.

Here are some other general guidelines:

- Test model performance with a small subset of your dataset (e.g., 20 images) to quickly assess learning progress by monitoring if validation loss decreases and the metrics on the validation set are increasing. extend the training to the full dataset.

- Manage memory (either RAM or video memory) by starting with an input resolution of around 500 x 500 pixels. The resolution can be incrementally increased.
- Carefully select the learning rate, as it significantly impacts model training. A learning rate that is too large may cause the model to diverge or produce unstable results. For example, using a learning rate of 0.1 on our point dataset caused the loss to become null, resulting in training failure. Conversely, a very small learning rate can result in slow learning and require a large number of epochs to converge. We recommend that users try multiple training runs with different learning rates and monitor performance to find an appropriate setting for their dataset.
- Better performance may be achieved by increasing the input resolution, training set size, training epochs, and training level. Increasing these settings leads to longer training times. Results from runs with various configurations are provided in the Supplementary Material, where some performance differences can be observed across settings. However, we note that these comparisons are based on a small number of runs and should be interpreted with caution.

Users can change these settings to fit their datasets and research requirements.

Discussion

In summary, PhenoLearn provides a user-friendly, high-throughput data extraction pipeline with fully integrated GUIs, enabling biologists without extensive computational skills to effectively measure phenotypic traits from images. While tools like DeepLabCut and Argos offer robust solutions for specific phenotyping tasks, they focus more deeply on animal tracking, primarily supporting point-based annotations. In contrast, PhenoLearn combines support for point annotations and segmentation tasks within a single toolkit and has already been successfully applied for both annotation types in previously published studies (Cooney et al., 2022; Y. He et al., 2022, 2023). PhenoLearn also includes functions tailored specifically for handling 2D image datasets of natural history collections. These features include 'Fast Labelling,' which streamlines the annotation naming process, and 'Review Mode' and 'Review Assistant,' which leverage specimen metadata to simplify the review process. These capabilities make PhenoLearn particularly suited for natural history collections, which often include rich metadata. Together, these features position PhenoLearn as a complementary tool for phenotyping 2D images, offering unique advantages for researchers working with such datasets.

As Lürig (2022) highlights, classic computer vision methods are more accessible to biologists with only CPUs. To facilitate the wider application of deep learning among biologists without GPU access, PhenoLearn leverages pre-trained models and partial model training to shorten CPU training times. Moreover, small training sets can yield accurate predictions for photographs with a highly consistent digitisation set-up, as minimal variation among images may bring more efficient training (Mulqueeney et al., 2024). From our results, it appears that CPU usage requires slightly more memory compared to GPU usage. However, it is more cost-effective to upgrade system RAM than to purchase GPUs with equivalent VRAM capacity. Additionally, most current consumer-grade laptops are equipped with at least 8 GB of RAM, making it feasible for a wide range of researchers to run PhenoLearn effectively on readily available CPU hardware. These features make predicting annotations on digitised specimens possible using only CPUs.

The modular design of PhenoLearn, comprising separate modules for image annotation (PhenoLabel) and deep learning (PhenoTrain), offers flexibility to integrate with other tools. This feature is particularly important in the fast-developing field of machine learning, where new and powerful methods are continually being developed such as Segment Anything (Kirillov et al., 2023), the foundation model for semantic segmentation. Thus, with PhenoLearn, users have the option to export annotations from PhenoLabel for other Deep Learning methods, and then use PhenoLabel again for efficient prediction reviewing. PhenoLearn supports multiple output formats (CSV, JSON, and image-based segmentation), making it compatible with other methods or toolkits. These formats can be easily converted into target Python data structures commonly used in deep learning pipelines. For example, regardless of the format, annotations can be transformed into 2D tensors that represent segmentations or point heatmaps, which are among the most used data structures for segmentation and point predictions. PhenoLabel can also simply serve as a manual labelling tool for small datasets.

Taken together, PhenoLearn is a versatile toolkit that bridges the gap between biological image datasets and downstream analysis, facilitating greater access for researchers to deep learning tools for image processing and data extraction.

Future Directions

Future development of PhenoLearn will likely focus on four main areas: (1) Optimisation of the user interface based on user feedback to increase usability. (2) Improvement of software performance, such as integrating multi-threading for displaying thumbnails, which will increase the efficiency of the review process. (3) Expansion of supported annotation types based on future user requirements. Adding bounding box annotations, for instance, could significantly broaden the toolkit's applications, including object recognition tasks which can be used to identify specimen appearances in laboratory or camera trap photographs. (4) Integrating alternative and newer models, such as Segment Anything (Kirillov et al., 2023) and other state-of-the-art deep learning models, to further enhance segmentation and landmark prediction capabilities.

References

Adams, D. C., & Otárola-Castillo, E. (2013). geomorph: An R package for the collection and analysis of geometric morphometric shape data. *Methods in Ecology and Evolution*, *4*(4), 393–399.

Blagoderov, V., Kitching, I. J., Livermore, L., Simonsen, T. J., & Smith, V. S. (2012). No specimen left behind: Industrial scale digitization of natural history collections. *ZooKeys*, *209*, 133.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Chang, J., & Alfaro, M. E. (2016). Crowdsourced geometric morphometrics enable rapid large-scale collection and analysis of phenotypic data. *Methods in Ecology and Evolution*, *7*(4), 472–482. https://doi.org/10.1111/2041-210X.12508

Chen, L.-C., Papandreou, G., Schroff, F., & Adam, H. (2017). *Rethinking Atrous Convolution for Semantic Image Segmentation* (arXiv:1706.05587). arXiv. http://arxiv.org/abs/1706.05587

Cooney, C. R., He, Y., Varley, Z. K., Nouri, L. O., Moody, C. J. A., Jardine, M. D., Liker, A., Székely, T., & Thomas, G. H. (2022). Latitudinal gradients in avian colourfulness. *Nature Ecology & Evolution*, *6*(5), 622–629. https://doi.org/10.1038/s41559-022-01714-1

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988. https://doi.org/10.1109/ICCV.2017.322

He, Y., Cooney, C. R., Maddock, S., & Thomas, G. H. (2023). Using pose estimation to identify regions and points on natural history specimens. *PLOS Computational Biology*, *19*(2), e1010933. https://doi.org/10.1371/journal.pcbi.1010933

He, Y., Varley, Z. K., Nouri, L. O., Moody, C. J. A., Jardine, M. D., Maddock, S., Thomas, G. H., & Cooney, C. R. (2022). Deep learning image segmentation reveals patterns of UV reflectance evolution in passerine birds. *Nature Communications*, *13*(1), 5068. https://doi.org/10.1038/s41467-022-32586-5

Huh, M., Agrawal, P., & Efros, A. A. (2016). *What makes ImageNet good for transfer learning?* (arXiv:1608.08614). arXiv. https://doi.org/10.48550/arXiv.1608.08614

John, A., Theobald, E. J., Cristea, N., Tan, A., & Hille Ris Lambers, J. (2024). Using photographs and deep neural networks to understand flowering phenology and diversity in mountain meadows. *Remote Sensing in Ecology and Conservation*, *10*(4), 480–499. https://doi.org/10.1002/rse2.382

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., & Girshick, R. (2023). *Segment Anything* (arXiv:2304.02643). arXiv. http://arxiv.org/abs/2304.02643

Lin, T.-Y., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *CoRR*, *abs/1405.0312*. http://arxiv.org/abs/1405.0312

Lürig, M. D. (2022). *phenopype*: A phenotyping pipeline for Python. *Methods in Ecology and Evolution*, *13*(3), 569–576. https://doi.org/10.1111/2041-210X.13771

Maia, R., Gruson, H., Endler, J. A., & White, T. E. (2019). pavo 2: New tools for the spectral and spatial analysis of colour in r. *Methods in Ecology and Evolution*, *10*(7), 1097–1107. https://doi.org/10.1111/2041-210X.13174

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., & Bethge, M. (2018). DeepLabCut: Markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, *21*(9), 1281–1289. https://doi.org/10.1038/s41593-018-0209-y

Mulqueeney, J. M., Searle-Barnes, A., Brombacher, A., Sweeney, M., Goswami, A., & Ezard, T. H. G. (2024). How many specimens make a sufficient training set for automated three-dimensional feature extraction? *Royal Society Open Science*, *11*(6), rsos.240113. https://doi.org/10.1098/rsos.240113

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).

http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pennekamp, F., & Schtickzelle, N. (2013). Implementing image analysis in laboratory-based experimental systems for ecology and evolution: A hands-on guide. *Methods in Ecology and Evolution*, *4*(5), 483–492.

Porto, A., & Voje, K. L. (2020). ML-morph: A fast, accurate and general approach for automated detection and landmarking of biological structures in images. *Methods in Ecology and Evolution*, *11*(4), 500–512. https://doi.org/10.1111/2041-210X.13373

Ravinet, M., Westram, A., Johannesson, K., Butlin, R., André, C., & Panova, M. (2016). Shared and nonshared genomic divergence in parallel ecotypes of L ittorina saxatilis at a local scale. *Molecular Ecology*, *25*(1), 287–305.

Ray, S., & Stopfer, M. A. (2022). Argos: A toolkit for tracking multiple animals in complex visual environments. *Methods in Ecology and Evolution*, *13*(3), 585–595. https://doi.org/10.1111/2041-210X.13776

Rohlf, F. J. (2006). tpsDig, version 2.10. *Http://Life. Bio. Sunysb. Edu/Morph/Index. Html*.

Schwartz, S. T., & Alfaro, M. E. (2021). *Sashimi*: A toolkit for facilitating high-throughput organismal image segmentation using deep learning. *Methods in Ecology and Evolution*, *12*(12), 2341–2354. https://doi.org/10.1111/2041-210X.13712

Shedrawi, G., Magron, F., Vigga, B., Bosserelle, P., Gislard, S., Halford, A. R., Tiitii, S., Fepuleai, F., Molai, C., Rota, M., Jalam, S., Fatongiatau, V., Sami, A. P., Nikiari, B., Sokach, A. H. M., Joy, L. A., Li, O., Steenbergen, D. J., & Andrew, N. L. (2024). Leveraging deep learning and computer vision technologies to enhance management of coastal fisheries in the Pacific region. *Scientific Reports*, *14*(1), 20915. https://doi.org/10.1038/s41598-024-71763-y

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning. In V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, & I. Maglogiannis (Eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018* (Vol. 11141, pp. 270–279). Springer International Publishing. https://doi.org/10.1007/978-3-030-01424-7_27

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, *27*.

Zelditch, M. L., Swiderski, D. L., Sheets, H. D., & Fink, W. L. (2004). Geometric morphometrics for biologists: A primer. *Elsevier*, 457. https://doi.org/10.1016/B978-0-12-386903-6.00001-0