Revised MS: MEWC: A user-friendly AI workflow for customised wildlife-image classification

Preprint on *EcoEvoRXiv*

v2 was the version reviewed:
https://ecoevorxiv.org/repository/object/6405/download/14621/

v3 is the revised version referred to herein:
https://ecoevorxiv.org/repository/object/6405/download/17197/

We thank the editor (Cédric Sueur) and reviewers for their feedback. Minor revisions were recommended. A revised version has been uploaded to EcoEvoRxiv.

Below is an explanation of the changes made.

Reviewer #1 (Anonymous)

1. Comment on "Train Base vs Train Final": If I train just once, my base model **is** my final model. The paper doesn't make clear what the distinction means.

Response

Thank you for flagging the ambiguity. We have clarified the Methods paragraph that introduces classifier training (L189-200) and added an explicit call-out to Figure 1. The revised text reads:

*"The Classifier Training phase starts with the '**Train Base**' stage (see Fig. 1) for initial fast, supervised training of only the top DenseNet layers (e.g., a compression layer, dropout layer, and a classifier layer of fully connected neurons), set up in a sequential DNN model with a frozen pre-trained ImageNet (Deng et al. 2009) or custom model base (e.g., an existing model of the same architecture hosted on the HuggingFace model repository[1]). This allows the user to take advantage of transfer learning for lower-level features (Huh et al. 2016). Once stabilised, the system can then proceed with the '**Train Final**' stage, where the user fine-tunes a partially (or fully) unfrozen model, with the number of DNN blocks unfrozen being dependent on how much new data is available for re-training and fine-tuning. This approach also means that new expert-identified snips can be easily incorporated into the workflow, to improve upon existing pre-trained classifier models progressively using multiple stages of transfer learning. We demonstrate the features of this approach using an example data set from our own work."*

[1] https://huggingface.co/bwbrook/mewc_pretrained

In short, “Train Base” = *initialise the task-specific head*, whereas “Train Final” = *fine-tune selected backbone layers*. Retaining both stages in the figure helps new users understand why MEWC sometimes makes two passes over the data, yet the orchestration script still allows a one-off “fast-and-done” training run for projects that do not require fine-tuning, or lack sufficient new training data to make that feasible.

2. Update EcoAssist to AddaxAI.

Every mention of “EcoAssist” has been replaced with “AddaxAI” (with a clarification of the name change in the appropriate place in the Methods) in both the text and figure captions, and the footnote now cites the new project URL.

3. Boxes are hard to see in Figure 3; consider re-rendering.

Figure 3 and Figure 4 have been re-rendered with 3-pixel red lines; boxes are now legible when printed or viewed in dark mode. The updated files are in the revised manuscript.

4. Various line edits and grammatical corrections:

All the line-level corrections flagged by Reviewer 1 have been implemented -- thank you for picking these up.

Reviewer #2 (Timm Haucke)

1. Clarify that MegaDetector fine-tuning is beyond the scope of MEWC, and users need to be aware of its limitations.

Modified/added (L144-169):

"*Detect calls MegaDetector v5a[2], a YOLO-based, open-source model trained on millions of annotated camera-trap images (Beery et al. 2019). For every image it (i) predicts tight bounding boxes, and (ii) assigns one of four coarse labels: animal, blank, human, or vehicle. MEWC writes the results into parallel sub-directories (*`animal/`, `blank/`, `human/`, `vehicle/`*) inside each camera-site folder; only the animal set proceeds to the next stage.*

[2] https://github.com/agentmorris/MegaDetector

Handling blanks. Any image routed to the `blank/` directory is still available to the user: nothing is deleted. If the project team suspects that MegaDetector has missed animals, they can simply drag-and-drop selected images or the entire contents of `blank/` into Camelot or AddaxAI, where the normal GUI tools make it easy to scroll, tag and recover any "false blanks." Once relabelled, those frames can be re-imported into MEWC for the next training round.

Snip crops each animal box and resizes it to 600 × 600 px (up- or down-scaling as needed). Cropping removes background cues and thereby reduces site-specific bias during training and inference.

Detector flexibility. MegaDetector v5a covers most "horizontal-view" deployments, but recall can drop in unusual geometries (steep top-down, fish-eye lenses, thermal-only). MEWC therefore treats the detector as a pluggable module. Users have three escalation options:

1. *Adjust the threshold. The confidence cut-off (`MD_CONF`, default = 0.20) is set in the YAML file. Lowering the value raises recall at the expense of more false boxes, which the downstream classifier's Blank class (if it has been included in the training data set) can absorb.*
2. *Fine-tune MegaDetector. Train on a small, locally-annotated bounding-box set and point the `DETECT_WEIGHTS` environment variable to the new `.pt` file.*
3. *Swap detectors entirely. Provide any model (e.g. Mask-RCNN, YOLO-v8) that emits COCO-style JSON or YOLO-txt; edit one line in `detect.sh` to call the new binary.*

Because the later snip–train–predict stages operate only on the cropped regions they receive, no further changes are necessary once a replacement detector is supplied. In short, MEWC depends not on MegaDetector itself but on some object detector that achieves adequate recall on the user's imagery."

2. Clarify choice of MegaDetector threshold:

We have added an explanation in Methods (L159-161) clarifying that MEWC ships with MegaDetector v5a's default threshold is 0.20 (given in response 1, immediately above).

Based on MegaDetector documentation, at that value the model achieves > 95 % recall while keeping the false-box rate manageable. The threshold is exposed in the YAML file as MD_CONF, so users can raise it (for faster, cleaner runs) or lower it (for maximum recall) with a one-line edit.

3. How are blanks handled and how does this relate to the MegaDetector threshold?

The revised paragraph cited above in response to comment #1 now more clearly explains the approach. In short:

1. MegaDetector sends any frame it labels *blank* directly to a blank/ directory; those images are not cropped or classified further.

2. If lowering MD_CONF still lets through false boxes, the default classifier includes a *Blank* class that captures them during inference. Projects that do not want that extra class can remove it from class_list.yaml and instead rely on a stricter threshold. These options are now described explicitly for users.

4. This paper [1] might provide some additional motivation for cropping before classifying.

Unfortunately, the link to the paper didn't come through.

5. 'Segmentation' term misleading in this context, use 'object detection'.

Duly noted and corrected.

6. Clarify how new models (e.g., via kimm API) can be added.

We have clarified this in the Methods (lines 181-188).

Beyond the 16 DNN model options already implemented in MEWC v2.0.0 (the currently most up-to-date version, using CUDA 12.3, cuDNN 8.9, TensorFlow 2.16.1, Keras 3.3.3 and JAX 0.4.28)[3], a wide range of other pre-trained image models are available via Python's Keras-Tensorflow libraries[4], as well as other frameworks like PyTorch[5]; MEWC can be readily adapted to use these as a replacement. Because the backbone is chosen via a single MODEL= entry in the YAML / env-file, any architecture registered in kimm (or in another external provider such as keras or timm) can be substituted by installing the new Python dependency and adding an API call in the model-selection function of the code. The user can then rebuild the Docker image with one line (`docker build -t mewc-train`*).*

7. In Table 2, add class-specific metrics.

This would blow out the size of the table in the MS, which is only meant to be indicative of the relationship between test accuracy and model scale, rather than class-specific metrics. However, exactly what the reviewer is looking for (the micro- and macro-averaged precision per class) is provided as part of the default output of MEWC Train, and so the users will always see this when using the software.

Added (Table 2 footnote):

---

[3] https://github.com/zaandahl/mewc-flow
[4] https://www.tensorflow.org/guide/keras
[5] https://pytorch.org

"*Note: Class-specific micro- and macro-averaged accuracy, precision and recall, are reported within MEWC Train*."

8. Few studies have 500K images per month!

True, such numbers are not the norm in wildlife studies. Revised to (L513-518):

"*For example, for-profit consultancies typically charge 1 cent per image processed (irrespective of animal or blank)*[6]*, which for a large-scale camera study amassing 100 thousand (K) images monthly, can escalate to an annual expenditure in the range of >$10K. For comparison, MEWC can operate on a mid-range dedicated GPU-enabled desktop PC with a one-off cost of ~$3K and electricity of <$0.5K yearly. The payback period under this scenario is brief (4–6 months).*"

9. Can MEWC work with OCI rather than just Docker?

MEWC is distributed as standard OCI-compliant images. We build them with Docker because that's the most familiar CLI, but the resulting artefacts run unchanged under any runtime that understands the OCI Image and Runtime specs, e.g., Podman, containerd/nerdctl, CRI-O, Apptainer/Singularity, Charliecloud, etc. On HPC systems that expose only Singularity/Apptainer the user can simply do

apptainer run docker://zaandahl/mewc-detect ... *or pull once and convert to a local *.sif file. No code changes are required.

Added (P4) a clarifying footnote (#2) to this effect.

[6] A $ figure like this can be obtained by requesting quotes for bulk image processing from various commercial operations; however, our intention here is not to target specific companies, so we have not cited any specifics.