Title

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

Barry W. Brook[1,*,a], Jessie C. Buettel[1,b], Peter van Lunteren[2,c] , Prakash P. Rajmohan[3,d], and R. Zach Aandahl[1,e]

[1] *School of Natural Sciences and the ARC Centre of Excellence for Australian Biodiversity and Heritage, University of Tasmania, Hobart, 7001, Tasmania, Australia.*

[2] *Addax Data Science, Utrecht, Utrecht, the Netherlands.*

[3] *School of Electrical Engineering and Computer Science, The University of Queensland, St Lucia, 4072, Queensland, Australia.*

*Corresponding email: barry.brook@utas.edu.au

[a] https://orcid.org/0000-0002-2491-1517

[b] https://orcid.org/0000-0001-6737-7468

[c] https://orcid.org/0000-0001-5488-4225

[d] https://orcid.org/0009-0006-6468-6314

[e] https://orcid.org/0000-0002-9412-8288

Abstract

Monitoring wildlife is crucial for making informed conservation and land-management decisions. Remotely triggered cameras are widely used for this purpose, but the resulting 'big data' are laborious to process. Although artificial intelligence (AI) offers a powerful solution to this bottleneck, it has been challenging for ecologists and practitioners without substantial technical expertise to tailor current approaches to their specific use cases. Generic, online offerings also have issues of ongoing costs and data privacy. Here we present an open-source, scalable, modular, cross-platform workflow, deployed using Docker containers, which leverages deep learning for wildlife image classification. It can be run using simple command-line prompts or via a user-friendly graphical user interface (AddaxAI). It enables end-users to easily execute a full range of tasks—from animal detection and counting to species identification—on local or cloud GPU-accelerated machines. It also integrates with the widely used open-source camera-trapping software 'Camelot', writing AI-classification data directly to image metadata and to CSV files, ready for either expert verification or direct data analysis. The result is a user-friendly but powerful multi-platform application for wildlife-image classification and research pipelines. An example case study with Tasmanian wildlife demonstrates the utility of our classifier training and inference workflow.

Keywords

Wildlife Monitoring; Artificial Intelligence; Image Classification; Deep Learning; Camera Trapping; Open-Source Software; Ecological Data Processing; Docker; MegaDetector; AddaxAI; Camelot

Introduction

Monitoring wildlife populations is crucial for research on population and community dynamics, habitat occupancy and suitability, conservation recovery and management, and control of invasive species (Pollock et al. 2002). An increasingly used tool for this purpose is the 'camera trap', a camera designed to be triggered based on motion or time that can be deployed for long periods in harsh environments. The number of studies using these remotely triggered cameras has increased rapidly, with one review documenting coverage at over 160,000 unique camera sites globally (Steenweg et al. 2017). However, this increase in camera-trap use has also led to data-handling constraints (Borowiec et al. 2022). A typical study involving a network of dozens to hundreds of field cameras, deployed for months at a time, can generate huge volumes of imagery (and sometimes video), which must be sorted and labelled before it can be used for ecological inference (Greenberg et al. 2019b).

Two main bottlenecks occur during image processing. First, false triggers (e.g., blank images caused by wind-blown vegetation or light reflections) and unwanted 'by-catch' (e.g., human hikers, vehicles, etc.) must be separated from the animal detections. Second, the animal images must be classified by a human, usually to species level. Both steps in this 'hand labelling' approach are tedious, repetitive, and prone to operator error, particularly due to fatigue (Falzon et al. 2020). It is also costly, in time and financial terms: properly trained personnel are a scarce and valuable resource. In this scenario data management, rather than data collection, becomes the limiting factor in the completion of research projects (Bubnicki et al. 2016). Despite rapid data acquisition in monitoring efforts, the cataloguing and classification frequently lag, leaving many datasets either unprocessed or underutilized (Young et al. 2018). This inefficiency not only risks data loss, detrimental to both science and conservation management, but also leads to limited applications, such as searching solely for target species while overlooking others. Given that the application of camera trapping as the primary field-based approach for vertebrate monitoring is set to continue—and indeed expand in the environments and spatial extents sampled—the demand for solutions to these problems is high (Boitani 2016). In short, there is a need for the data-processing workflow for camera traps to be reliable, cost effective and easy to use.

Recently, the information-technology community has been working with ecological data to develop applied artificial intelligence (AI) methods for wildlife-image processing (Nguyen et al. 2017). Building on the outstanding success of deep-learning approaches to consumer-based computer-vision applications (Nguyen et al. 2017, Christin et al. 2019), a range of AI solutions are now available for detecting and classifying wildlife (Tabak et al. 2019). For object localisation—finding and counting animals against complex backgrounds—this includes the open source 'MegaDetector', developed by Microsoft's AI Earth team (Beery et al. 2019), or customisable tools like 'Sherlock' (Penn et al. 2024). For automated species classification—providing suggested labels prior to final human-expert verification—one of the most widely used options is the non-profit web portal 'Wildlife Insights' (Ahumada et al. 2020). A plethora of private consultancy companies now also offer AI-assisted data-identification services, and there are also many classifiers published in the scientific literature (e.g., Falzon et al. 2020), with their code available for download.

However, the above-described image-processing options offer solutions that are incomplete or challenging to implement for most wildlife ecologists and practitioners (Young et al. 2018, Vélez et al. 2023). The commercial offerings are easy to use, but obviously involve ongoing costs, either via a subscription model or pay-per-image data processing, and the data is hosted by a third-party provider, which can raise legal questions about whom owns the uploaded data. This can also result in

an accumulation of significant ongoing costs when there is a high-volume throughput. Further, because they service a general (regional or global) customer base, the local fidelity of their classification models is typically constrained by a lack of specific training data (Schneider et al. 2020). This problem worsens if human-labelled data are used automatically to train the underlying classifier, without prior quality-control checks on the model-input data. In addition, web-hosted services like Wildlife Insights requires high-speed internet connectivity to undertake the expert classifications, and it uses global AI models that typically include many irrelevant species (i.e., those not found in the study area). Consultancies can involve long wait times for data transfer and processing, and a high per-unit cost. However, the alternative—customising and fitting in-house deep-learning models using a public code base—currently requires a level of expertise in programming and a familiarity with the complexities of computer vision that few ecologists or conservation practitioners possess.

Here we present a solution that breaks through these constraints by using a free, open-source, and easy-to-implement wildlife-classification workflow, suitable for use by ecologists, conservation managers, and citizen scientists in any system or environmental context. Our goals were to:

i) Leverage cutting-edge developments in computer vision, but leave the details behind the scenes, and use a system-agnostic distribution platform (Docker[1,2]) for deployment.
ii) Implement a scalable, modular, "code-free" image-data preparation and AI-model-training pipeline, controlled by a simple command-line interface.
iii) Make the workflow easy for non-specialists to use, completely reproducible, and flexible for expert-level fine-tuning or expansion.
iv) Seamlessly integrate the AI classification with existing, free applications such as the wildlife-image-database Camelot[3] and the graphical user interface AddaxAI[4], to streamline model deployment, label verification and data analysis.
v) Demonstrate both local (desktop PC) and cloud-compute (virtual machine) options, in both cases giving the user full control over their data privacy.

Our core advance is the creation of a user-friendly workflow that reframes deep-learning AI for wildlife-image classification as being something comfortably within the reach of ecologists, natural-resource managers, and citizen scientists. The use of Docker images allows for a significant layer of abstraction that removes much of the technical overhead usually required to implement deep-learning solutions for wildlife classification. This overturns the perception by many ecologists that it is an 'arcane art' needing specialist data-science and programming skills to implement (even if the underlying motivations for AI deployment are understood). We demonstrate the application of the workflow using a diverse labelled dataset of wildlife images from Tasmania.

[1] https://www.docker.com

[2] Note that MEWC is distributed as standard Open Container Initiative (OCI)-compliant images. We build them with Docker because that is the most familiar command-line interface, 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.

[3] https://gitlab.com/camelot-project/camelot

[4] https://addaxdatascience.com/addaxai

Methods

Implementing deep-learning systems for classifying wildlife images can be complex, technical, and time consuming to set up, especially for those who use camera-trap data but lack specialised expertise and in-depth knowledge of data-science principles, Python programming, and graphics processor unit (GPU) library functions (Nguyen et al. 2017, Tabak et al. 2019). The new approach we detail below is therefore useful because it allows non-experts to develop customised species classifiers within an intuitive, user-friendly, modular, and easily deployable workflow. In brief, it lets ecologists undertake all development stages within one, high-level framework: detecting individual animals from images, training an AI classifier model, doing the bulk classification tasks, facilitating expert review, and predicting species (or other) classes on new camera-trap data.

The design goal in developing this system was to ensure that it was easy to learn and apply, and flexible such that it could generalise to a wide range of contexts. Further, the system requirements and configuration steps are much simpler than typical for this type of project, such that they can be set up on new (or virtual) machines with GPU acceleration, by a non-IT expert, in a short space of time. To do this, the pipeline for image-processing and classification is built around the Docker Container engine, using modern software-engineering techniques and consists of a repeatable series of steps, with all assets (code, scripts, etc.) packaged for seamless 'behind the scenes' distribution.

(i) Development of the classifier and inference pipelines

There are three main stages to the workflow, each implemented as a Docker module (details in section below, and Fig. 1 schematic): animal detection, classifier training and field-service prediction. We call this framework 'MEWC' (the Mega-Efficient Wildlife Classifier): a playful riff on the use of MegaDetector for object detection and the EfficientNet deep-learning model as the default for animal-image classification. The entire pipeline can be run in sequence, or any step can be run as a separate module, depending on the use case. For example, if a trained model is already available and one simply wants to classify a new tranche of field data, the Classifier Training step can be skipped.

Animal Detection: MEWC runs two sequential steps, 'Detect' then 'Snip'.

Detect calls MegaDetector v5a[5], 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.

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.

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

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.

Classifier Training: This is the approach to fitting and testing the AI wildlife-species classifier model.

We use supervised training of a deep convolution neural network model (LeCun and Bengio 1995, Krizhevsky et al. 2012) with the option to alternatively use a Vision Transformer (ViT), which we will collectively refer to as Deep Neural Network models (DNN). By default, we offer a wide range of alternative DNN model architectures (EfficientNet v2, ConvNeXt, ViT) and sizes via the Keras 3 Image Model Zoo (kimm)[6] application programming interface. In general, the larger the model size and input scale of the images, the slower it is to train, the more compute (GPU, RAM) is required, and the more training examples it needs to process for optimal performance. The trade-off is, if all of these needs are met, the larger models typically yield higher generalisation accuracy, although they might require stronger regularisation regimes to avoid overfitting on smaller datasets (Chollet 2021).

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)[7], a wide range of other pre-trained image models are available via Python's Keras-Tensorflow libraries[8], as well as other frameworks like PyTorch[9]; 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`).

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

[6] https://github.com/james77777778/keras-image-models
[7] https://github.com/zaandahl/mewc-flow
[8] https://www.tensorflow.org/guide/keras
[9] https://pytorch.org

hosted on the HuggingFace model repository[10]). 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.

We include two forms of regularisation to mitigate over-fitting (Santos and Papa 2022): stochastic dropout is used to promote model generalisation by randomly setting a subset of outputs in a layer to zero during training, and random image augmentation techniques to increase the diversity of the training data (Xu et al. 2023). The augmentation techniques include cropping, zooming, flipping, adding noise, blurring, and adjusting brightness and contrast, as well as applying affine transformations. In the MEWC configuration file, formatted in YAML[11], these regularisation techniques can be staged across epochs. This allows for progressively intensification of the regularisation as the model training advances, particularly as the training loss decreases and the risk of over-fitting increases. Many other parameters can be specified in the YAML file which can be overridden by setting environment variables, as detailed in the online vignettes.

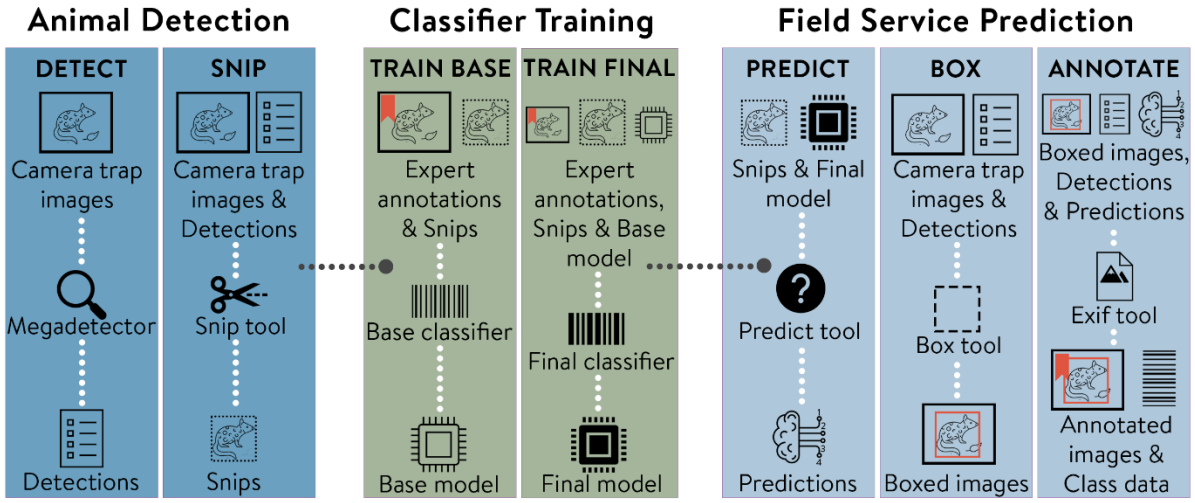


Figure 1. The MEWC workflow. Each step in the camera-trap-image processing pipeline has an associated Docker container. The Expert Annotations (species labels for the training images) provide the basis for building a supervised classification model (see Appendix details on this step). Detection data are combined with prediction data (inferred from the classifier) to produce the red-boxed images and final classification data. The classification data includes basic information written to metadata fields of the images, and a detailed CSV file (one image per row) that can be used for direct analyses or import into a dedicated camera-trap database like Camelot for rapid expert validation.

[10] https://huggingface.co/bwbrook/mewc_pretrained
[11] https://yaml.org

Field-Service Prediction: These steps are critical for the translation and interpretation of the classified images to their use and analysis. For this phase we have defined three chained steps: 'Predict', 'Box' and 'Annotate'. 'Predict' uses the trained classification model to output categorical predictions for each animal snip. 'Box' and 'Annotate' then draw coloured bounding boxes around animals on the original camera-trap images, and write model predictions to the EXIF image metadata, respectively. This prepares the images for import into camera-trap organisational software. Predict also writes all classification data to a .CSV file in a form suitable for analysis in, for example, R[12]. Prediction of species and post-processing capabilities are also available through the EcoAssist application.

(ii) Docker Containers for deployment and distribution

We use *Docker* to underpin the MEWC pipeline. Docker is a free-to-use, open-source software-management system that automates the deployment of applications inside portable containers (Miell and Sayers 2019). It is cross-platform and handles all the software and code dependencies automatically, which greatly simplifies to use of complex software-installation setups. It can be run using simple commands. Using Docker, all libraries, and drivers (such those requiring the NVIDIA CUDA development kit[13] for harnessing the power of GPUs), can be deployed as needed in the background, rather than requiring the user to follow a series of highly technical installation steps and compatibility checks. Docker allows applications to be distributed as downloadable Docker image files, hosted via Docker Hub, which are then instantiated and run locally as containers—lightweight software layers that contain all dependencies, code, and configuration required to run an application.

In practical terms, this means ecologists using our AI pipeline, whether for model training or just classification using a default model, only need to download and install a single application: Docker. For the command-line interface, the user can choose from PowerShell (Windows) or bash (Linux, Mac) to act as a simple, flexible, yet powerful and customisable interface to the Docker components. In its default form, MEWC via Docker requires the user to issue only a few simple, reproducible commands, to trigger the orchestration scripts, which then do all the required work for the user.

While each of the Docker containers (for Detection, Snipping, Training, Inference, and Prediction) are self-contained, a run of the entire pipeline can be orchestrated from a PowerShell or bash script. In this way a single script command iterates through a full field service consisting of multiple sites/cameras, identifies any animals in the images, reports classification predictions, and annotates the camera-trap images that contain animals by drawing bounding boxes and adding AI-prediction information. The end result of using this container system is that all a user (ecologist, natural-resource manager, citizen scientist) needs to do to create and run a sophisticated, customised deep-learning classifier is to: (i) set some configuration files (or accept the defaults), (ii) supply the training images in labelled folders and the path to these, and (iii) click-to-run a PowerShell or Bash script.

Both the code, developed in Python and PowerShell/Bash, and applications which are managed and distributed seamlessly by Docker, are made available for download and modification from publicly accessible repositories (GitHub for code: http://github.com/zaandahl/mewc, and a delivery-and-update mode for application containers via Docker Hub; see Appendix).

[12] https://www.r-project.org
[13] https://developer.nvidia.com/cuda-toolkit

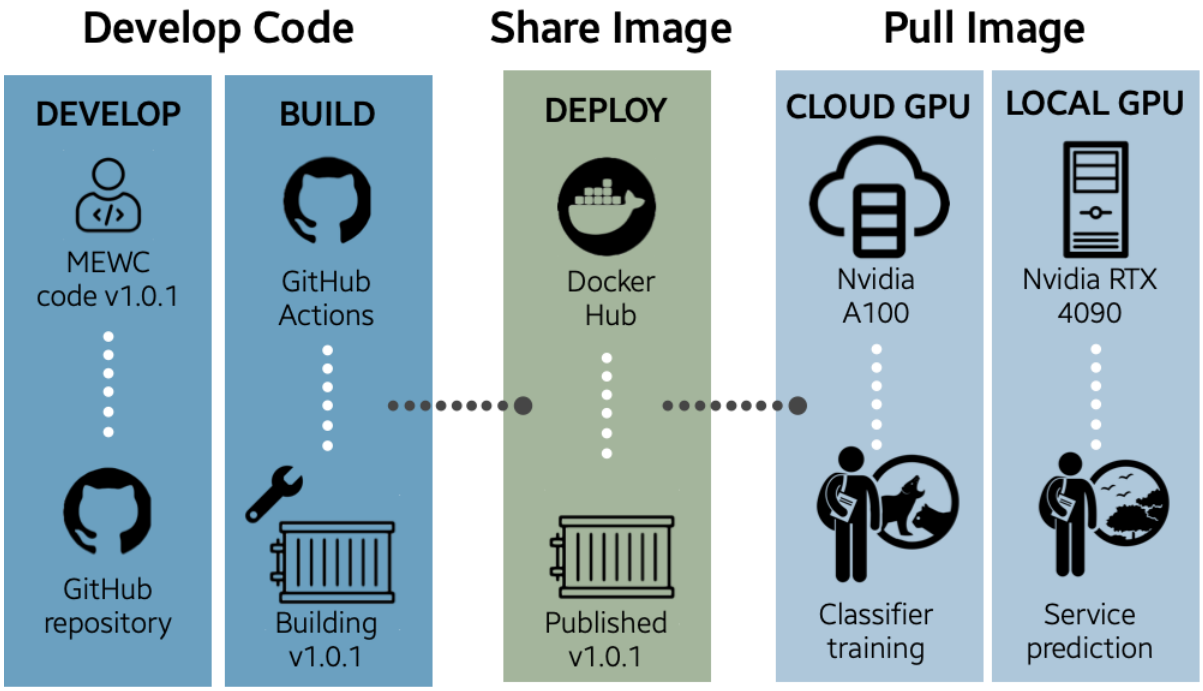


Figure 2. Workflow for the MEWC infrastructure pipeline, illustrating how this allows for a seamless deployment to a local or Virtual Machine (VM), for both model training and field-service inference. The process for a typical user is high-level and therefore very straightforward. Behind the scenes, the developer uses GitHub Actions as a continuous integration / continuous deployment pipeline for pushing code and version changes to GitHub, which are then drawn on to build an up-to-date Docker image. With new code updates (e.g., v1.0.1 in the example), the Docker image is pushed to DockerHub, where it is public and available to be pulled down by the user, to either a local GPU-enabled computer or a cloud-based resource (a VM) on a service such as Nectar (in Australia) or Amazon Web Services (commercial). From there, the user can undertake classifier training or service prediction. The code-as-infrastructure tools Terraform and Ansible are used to build and tear down remote-cloud environments as required, thereby minimising costs by ensuring that images are never left to idly consume compute resources. See text for specific definitions of packages and services.

We have also included the ability to run MEWC on a virtual machine (VM), to take advantage of high-performance cloud computing (Fig. 2). We used as our exemplar the Australian Research Data Commons 'Nectar' Research Cloud[14], but it would work with any similar government/research or pay-per-use commercial infrastructure (e.g., Azure, AWS, Lambda etc.). To deploy the MEWC pipeline, we use two Infrastructure-as-Code tools: (i) *Terraform*[15], to provision the VM and set up the networking, storage, etc., and then (ii) *Ansible*[16], to install Docker on the VM and pull the Docker images. Both tools are used in tandem to create a robust, automated deployment pipeline. In our vignettes (GitHub), we provide a full worked example from the user's perspective.

[14] https://ardc.edu.au/services/ardc-nectar-research-cloud
[15] https://www.terraform.io
[16] https://www.ansible.com

(iii) Integration of MEWC with the Camelot Database

Camelot is an open-source camera-trapping software tool for wildlife researchers and conservationists (Hendry and Mann 2018). Camelot serves as a widely used database for viewing and expert-tagging camera-trap photos, tracking site-species data, and facilitating preliminary data analyses. Its compatibility with various operating systems (Windows, MacOS, Linux) and its user-friendly interface makes it a favoured choice for integration with MEWC. Our solution exploits Camelot's ability to read and search image metadata tags, by writing the AI-classification information to each image's EXIF. This eliminates the need for any modifications to the Camelot software and streamlines the process of moving from AI-based classification to expert verification to data analysis.

In brief, once images are processed and labelled by the MEWC pipeline, the data from each service and site can be imported into Camelot as usual. The classifications are automatically read and can be recovered simply by searching within Camelot for the relevant metadata field. We chose three numerical EXIF fields for this purpose: `photo-iso-setting` for classification (represented by a unique integer for each species), `photo-exposure-value` for confidence, expressed as an integer from 01 to 99 (representing the relative percentage confidence that the image is of the target class), and `photo-fnumber-setting` for the number of MegaDetector animal detections (integer).

Importantly, this integration offers a value-added workflow that not only leverages the advanced machine-learning capabilities of MEWC but also benefits from the robust database management, initial analysis features, and output formatting of Camelot. For the user, this means a comprehensive, 'full-stack solution' for camera-trap-based wildlife monitoring, all within an open-source ecosystem.

(iv) Integration of MEWC with the AddaxAI application

AddaxAI (van Lunteren 2023) is Python-based application which is designed to enable automatic species identification within a graphical user interface. The package, previously called EcoAssist, allows the user to deploy MEWC classification models and post-process imagery without having to write code. Installation is automated and dependencies will be installed in virtual environments to avoid conflicts. Postprocessing features include folder separation, detection visualisation, cropping, label creation, and exporting results to CSV files. It will automatically run on NVIDIA or Apple Silicon GPU if detected and is available for Microsoft Windows, Apple macOS, and Linux. AddaxAI also offers an option to make custom models available for all users. This system integrates well with the popular image-tagging software Timelapse (Greenberg et al. 2019a).

(v) Case Study: Building a Classifier and Processing a Camera Service for Tasmanian Wildlife

We demonstrate the entire MEWC workflow using an example camera-trap dataset collected and curated by two of the authors (BWB and JCB) from Tasmania, Australia[17]. These images are drawn from a wide variety of environmental contexts (dry and wet temperate eucalypt forest, woodland, and grasslands) using white-, infra-red (IR) and no-glow flash types from Cuddeback, Reconyx, Swift and Bushnell cameras. The following species or aggregated classes are represented in the dataset: Tasmanian Pademelon (*Thylogale billardierii*), Bennetts Wallaby (*Notamacropus rufogriseus*),

[17] https://dx.doi.org/10.25959/wm5g-b990

Tasmanian Devil (*Sarcophilus harrisii*), Feral Cat (*Felis catus*), Bare-nosed Wombat (*Vombatus ursinus*), Brushtail Possum (*Trichosurus vulpecula*), Fallow Deer (*Dama dama*), Southern Brown Bandicoot (*Isoodon obesulus*), Currawong (Black: *Strepera fuliginosa*, Grey: *S. versicolor*) and Bronzewing (Brush: *Phaps elegans*, Common: *P. chalcoptera*). The latter two classes are birds, each of which consist of an aggregation of two species within a genus; the former eight are mammals. Images of each of these species is shown in Fig. 3, bounded by red detection boxes.

For implementing the classifier training, we provide 4 000 train and 1 000 test images for each of 10 different classes, for a total of 50 000 expert-labelled snips (each sized at 600- × 600-pixel, after being extracted from their original images using the MEWC-Snip tool). Then, for demonstrating the detection, inference, and post-processing pipelines (EXIF writing and image sorting), we provide a sequence of 100 images for each of four field cameras that were not used in training, located on the lead author's rural property in southern Tasmania: C3: IR flash, C7: white flash, C15: no-glow flash and C21: inbuilt IR flash. Other than the target wildlife, these images include some representations of blank images, humans (the lead author), and vehicles (trail-bike motorcycle), to demonstrate the four broad classes designated by the MegaDetector prior to classification on the animal images.

This case-study is not meant to be representative of most real-world cases (e.g., typically there will be more species to classify within a study region, and with an unbalanced number of images per species). It is sufficient, however, to showcase all the major features of the MEWC framework and can be used as a template for users to get a hands-on experience of classifier training. Moreover, it demonstrates that the default configuration settings for the classifier will produce robust results 'out of the box', whilst acknowledging that dedicated hyper-parameter tuning might further improve results. The training, test and inference data for the case study is available on the UTas Data repository, with a link on the GitHub site (see footnotes above).

(vi) Vignettes

On the MEWC GitHub site, we have included vignettes that guide users on how to implement a basic MEWC training and inference workflow, using the case-study data as the example. Other more detailed examples demonstrate how to: (a) use a labelled collection of snipped images (sorted into species folders) to train and deploy a customised image-classifier model, and (b) set up a VM and infrastructure in the ARDC Nectar Cloud. The training procedure on a cloud server are nearly identical to training on a local GPU machine set up as a VM with Linux (natively or with WSL[18]).

[18] https://learn.microsoft.com/en-us/windows/wsl

Figure 3. Images of wildlife from camera traps in Tasmania (the 10-species Case Study). Shown, left-to-right, are top: Tasmanian Devil (native predator), Feral Cat (introduced predator), Tasmanian Pademelon, Bennetts Wallaby and Bare-Nosed Wombat (native herbivores); bottom: Fallow Deer (introduced herbivore), Southern Brown Bandicoot and Brushtail Possum (native omnivores), Grey Currawong and Brush Bronzewing (native birds). Red bounding boxes are imprinted on the image after running the 'MegaDetector' wildlife detector. See main text for scientific names.

Results

Here we demonstrate the results of applying MEWC to the case-study dataset. This is only meant to be illustrative of the level of performance one might expect by using the system on a typical camera-trap dataset: in this case, a balanced dataset of 10 species and 50 000 labelled images. Models trained on smaller unbalanced datasets and/or more speciose communities (especially with closely related taxa and many rare species) are likely to be less reliable at generalising to new contexts, whereas those that can leverage even larger number of expert-classified images (e.g., from past efforts at manual classification) would likely yield even better outcomes than we report.

Classifier Training and Validation

We trained four DNN models (EfficientNet's B0, V2S, V2M and V2L) on two computer systems:

System **A** is a mid-range desktop machine-learning setup, now a few years out-of-date (as of 2024), but nonetheless typical of what many users might have available. It consists of 8 × i7-9700K CPU cores, 16 GB RAM and 2 × NVIDIA RTX 2080 GPUs, each with 8 GB of onboard RAM (these are the most meaningful specifications for deep learning). Note that when training the classifier or doing inference, MEWC can take advantage of multiple GPU run in parallel via hierarchical mirroring in TensorFlow. However, the other step that uses a GPU (detect) can only use one GPU per instance.

System **B** is hosted on a virtual machine (ARDC Nectar Cloud), with 64 × VCPUs (Intel Xenon Icelake), 128 GB RAM, and a single NVIDIA A100 GPU with 40 GB of on-board RAM.

The comparative results are detailed in Table 1. Mini-batch sizes for System A needed to be reduced from the baseline of 128 images as model size increased, due to GPU-RAM limitations, whereas for System B only the largest model required a reduction in mini-batch size. System B typically trained the models five to six times faster than System A, illustrating the value of having access to high-performance cloud infrastructure, with fewer computational bottlenecks. Yet even with the mid-range System A, a high-performing model could be trained on 50 000 images in about six hours.

Table 1. Training performance of the MEWC classifier on two different computer systems, using 40 000 images from the Case Study dataset. System A is a mid-range GPU-equipped desktop computer, and B is a high-performance cloud-based virtual machine. The initial convolutional neural network weights come from ImageNet pre-training, a form of transfer learning. Size (px) is the input image size in pixels (a square crop), Batch is the mini-batch size used during training (larger is faster but more memory intensive), Frozen is the time in seconds required for each epoch (a pass over the entire dataset) for the first stage of training when only the top classifier is trained to stabilisation and the base DNN model has its weights frozen (run for 5 to 10 epochs: a full pass of the data). Train is the second stage when the top two convolution blocks are unfrozen, such that these neurons can be trained at a low learning rate to fine-tune the final model weights. Best Epoch is the that which resulted in the lowest validation loss, and Total Training is the length of time required to complete the Frozen epochs and the Train epochs up to the Best Epoch.

DNN model				*Epoch time (s)*			
	Param	Size				Best	
A: 2 × RTX 2080 GPU[1]	(M)	(px)	Batch	Frozen	Train	Epoch	Total Training
EN-B0	5	224	128	108	133	42	1h 51m
EN-V2S	21	300	48	228	472	44	6h 24m
EN-V2M	54	384	32	589	951	35	10h 53m
EN-V2L	119	480	12	1331	3213	31	1d 7h 22m
B: A100 GPU[2]							
EN-B0	5	224	128	11	24	42	25m
EN-V2S	21	300	128	45	103	44	49m
EN-V2M	54	384	128	131	219	35	2h 52m
EN-V2L	119	480	64	351	594	31	4h 21m
ConvNeXt-Base	89	384	64	232	314	35	3h 42m
ViT-Base	87	384	64	215	275	38	3h 30m

[1]Alienware PC: 2 × NVIDIA RTX 2080 GPU (8 GB RAM each), 16 × i7 CPU, 16 GB system RAM

[2]ARDC Nectar Virtual Machine: NVIDIA A100 GPU (40 GB RAM), 64 VCPU, 128 GB RAM

As illustrated in Table 2, all models formed well on the 10 000 held-out test images of the case-study dataset. As one example, the moderately sized and fast-to-train EN-V2S, with an input-image size (snip) of 300 pixels, and 20.4 million parameters had its lowest validation loss after 44 epochs of training (an epoch is a full pass over all training data) and this resulted in a 99.48% classification accuracy based on the test data. The smaller EN-B0 model was underfit (as revealed by its lower test accuracy), whereas the additional representational power offered by the wider and deeper neural network of EN-V2L could not be sufficiently leveraged by a dataset the size of the case study (i.e., it was overfit, despite regularisation via stochastic dropout and random augmentations). This result underscores the utility of even modest-sized DNNs for the task of wildlife classification, and of

matching the model choice to the quantity and quality of the available information. That said, further tuning of the regularisation hyper-parameters would likely improve the results for all the models.

Note also that one of the fastest model options to train was EN-V2XL when only the final classification layer (the 'model top') was fitted to the case-study dataset, with all other DNN layers being left 'frozen' from their ImageNet baseline. On system B this could be trained for 5 epochs at 351 seconds per epoch, for a total training time of ~30 minutes, and yet this still yielded a test classification accuracy of 96.53 %. This illustrates the boost given by model pre-training and corresponding transfer learning. Wildlife-image classification need never start from scratch. But the result for V2S nonetheless underscores the value of fine-tuning the deeper DNN blocks.

Detection and Inference

Applying MEWC's detection-inference workflow to the example field cameras (4 × 100 images):

(i) MegaDetector correctly identified 394 animals, 16 blank images, 11 people and 3 vehicles. All animal images with more than one individual present were correctly located (26 images). There was 1 image falsely labelled as a blank, and 2 animals missed by the detector in two multi-animal images (the other target was correctly detected). In one case a person was misattributed to the animal class, because only the boots were detected. The sensitivity of MegaDetector can be changed in the YAML configuration file, acknowledging the trade-off between misses and false detections.

(ii) For the 394 valid animal detections, the 10-species EN-V2S made 6 errors (with never more than two errors per class), for an overall classification accuracy of 98.5% on the new camera-service data. Most errors were difficult 'edge cases', where neither a human expert nor a machine-learning model will do well due to insufficient image clarity, and this resulted in typically low classification probabilities for these images (Fig. 4). One image contained a species not represented in the training data (echidna, *Tachyglossus aculeatus*): this was labelled as a bronzewing with low probability (0.39). This shows that when input data are out-of-scope, the 'next best' classification will be chosen.

Table 3 gives an example of the model-proposed classification information for each image, written to a CSV file by the inference step. This can be useful for importing into ecological software (e.g., R script) for direct analysis. It is also used in the annotate step of MEWC, wherein the classification data are written to the image metadata. This metadata information is then available for searching within the Camelot camera-trap image management database, as illustrated in Fig. 5, wherein the user is offered a preliminary classification of each image, which can then be expert-verified. A similar approach to post-AI annotation can be taken with the AddaxAI application (Fig. 6).

444 **Table 2.** Test performance of each trained deep neural network model (see Table 1), using 10 000
445 images from the Case Study that were held back from model training. The number of parameters
446 (millions) and size (px) is the dimensions in pixels of the input image (cropped by the MegaDetector).
447 Test loss function is categorical focal loss (lower is better), and the test accuracy is the proportion of
448 images where the correct species (out of 10 possible candidates) was the top-ranked selection.

DNN model	Param (M)	Size (px)	Test Loss	Test acc.
EN-B0	5	224	0.0047	0.9894
EN-V2S	21	300	0.0023	0.9948
EN-V2M	54	384	0.0021	0.9952
EN-V2L	119	480	0.0024	0.9947
ConvNeXt-Base	89	384	0.0019	0.9963
ViT-Base	87	384	0.0018	0.9960
V2XL-Frozen	208	480	0.0139	0.9653

449

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

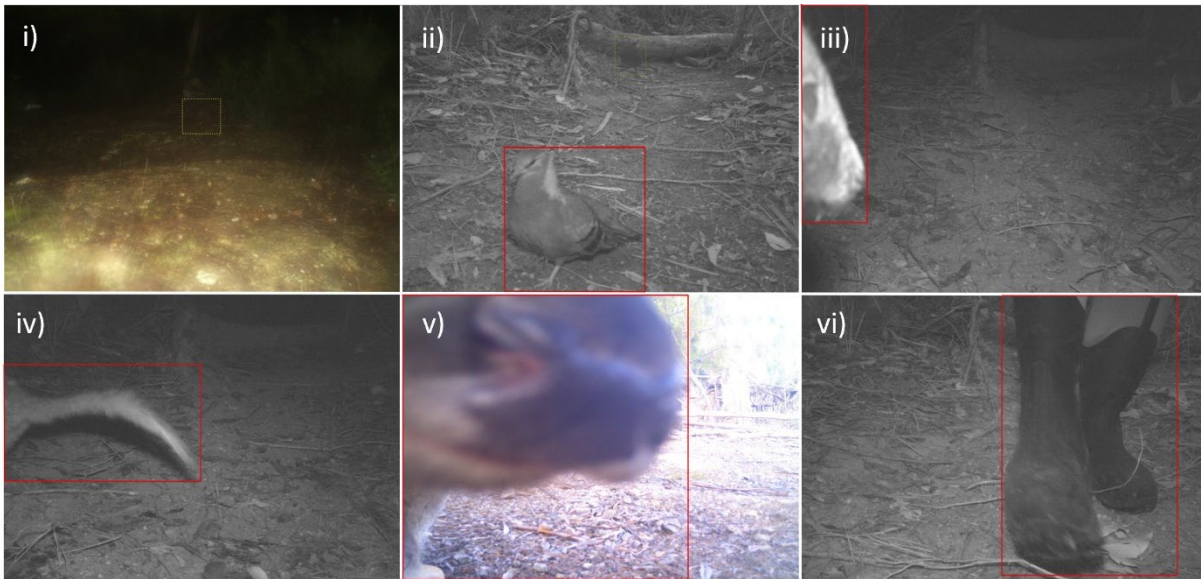
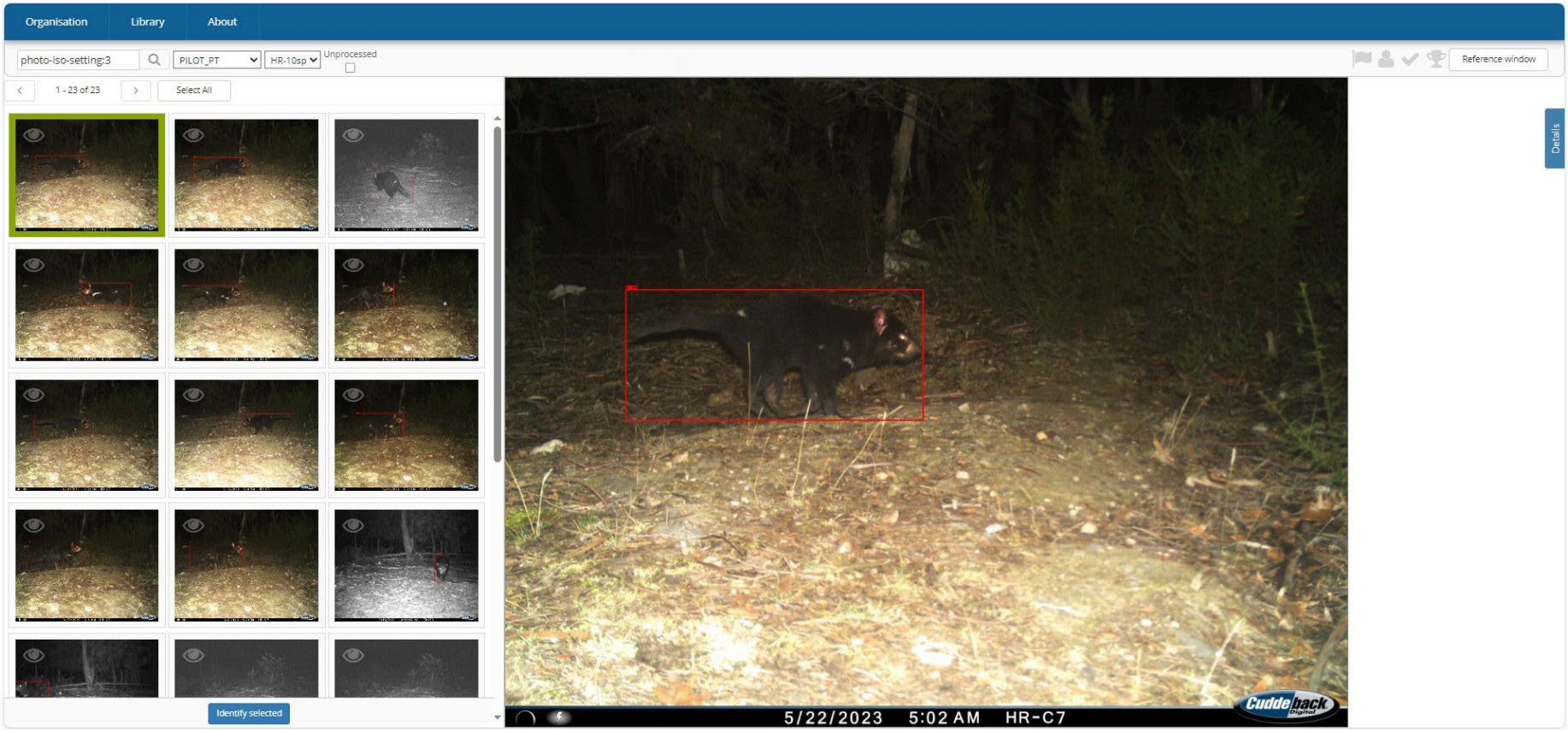


Figure 4. Examples of the few detector and classifier errors on the four field-camera datasets, which had 100 images per camera. Diagnosis (left to right, top to bottom): (i) MegaDetector failed to detect the Tasmanian pademelon (indicated with the green dashed box) due to foggy conditions. (ii) The front bird (a bronzewing) was detected, but the back one was not (green dashed box). (iii) The animal (a Tasmanian pademelon) was classified as a bare-nosed wombat, possibly because only the flat front of the face was detected. (iv) Misclassification, probably because the brushtail possum has alopecia on the tail, making it resemble that of a Tasmanian devil tail. (v) Misclassification, with the fallow deer nose being too close to the camera and appearing like a Bennett's wallaby nose. (vi) Boots worn by the person checking the camera, falsely labelled by MegaDetector as an animal, and then assigned by the species-classifier model as a currawong (a black-coloured bird) by the classifier.

Table 3. Example of the CSV file generated after running the detection, snip, and inference steps of MEWC, shown here for one of the evaluation dataset of field cameras (HR-C3). The subscript after the 'filename' allows for multiple snips per image (indexed from 0 and listed one per row). The 'class_id' is an integer classification written to the ISO field of the image's metadata, and 'prob' is the deep-learning model's probability of the classification (relative to all other possible classes), written to the metadata's Exposure field as a two-digit integer (e.g., 99). Also given is the 'class name' (specified, along with the class_id, in a configuration YAML file), a 'rand_name' assigned to each snip (random combination of ASCII characters, to allow for snip pooling for later use in classifier training, without duplicate filenames), the date-time of the original image 'date_time_orig', and the MegaDetector confidence of the bounding-box object 'conf'. Note that the user can specify whether to show only the top-ranked class, or to show the probabilities for all classes in the model (which provides for more fine-grained classification information, at the cost of a larger file size).

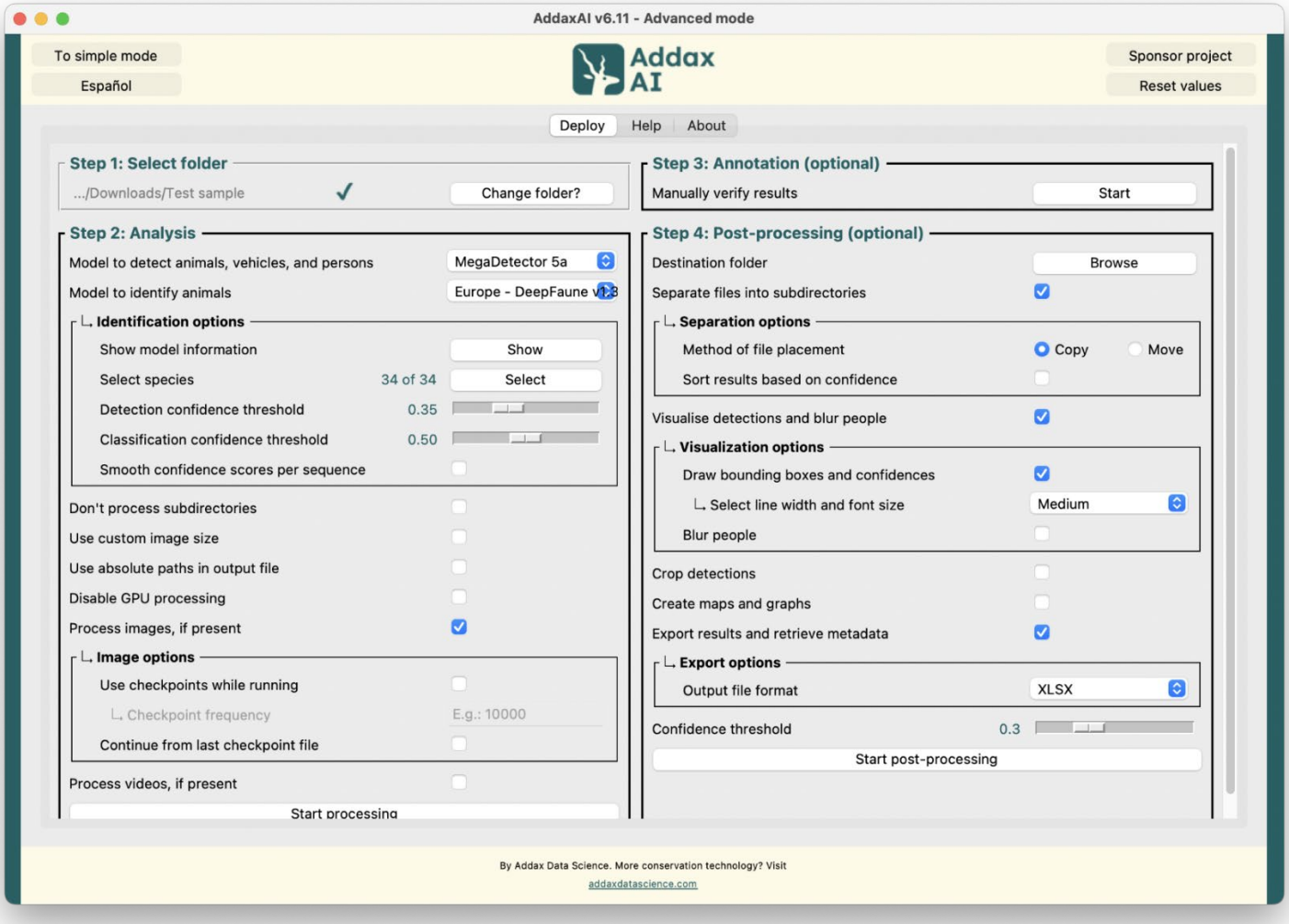
filename	class_id	prob	class_name	rand_name	date_time_orig	conf
hr_c3_1-0.JPG	1	0.990756	tasmanian_pademelon	Sz3E6hbA72Z4LvIq.JPG	2023:05:14 02:48:26	0.926
hr_c3_2-0.JPG	4	0.998375	brushtail_possum	8QF1G8NfJaoHBTNp.JPG	2023:05:14 18:26:48	0.934
hr_c3_3-0.JPG	1	0.992052	tasmanian_pademelon	izYy2Ok3oPDpf3sm.JPG	2023:05:16 05:05:12	0.947
hr_c3_4-0.JPG	1	0.996194	tasmanian_pademelon	g9vajIa26vFXrjXK.JPG	2023:05:16 17:50:40	0.881
hr_c3_5-0.JPG	1	0.914644	tasmanian_pademelon	evThO27qscaHh1TX.JPG	2023:05:17 04:04:12	0.792
hr_c3_6-0.JPG	2	0.998748	bennetts_wallaby	wAMGyn3AdrxFgLzO.JPG	2023:05:17 06:37:16	0.938
hr_c3_7-0.JPG	4	0.998264	brushtail_possum	RU1V5BmrruZ0S04b.JPG	2023:05:17 23:08:52	0.964
hr_c3_8-0.JPG	6	0.999578	bare_nosed_wombat	mjipciLeV28lEoBa.JPG	2023:05:18 00:05:26	0.891
hr_c3_9-0.JPG	4	0.999596	brushtail_possum	vkvqhVoL3FNosyne.JPG	2023:05:18 20:44:28	0.948

Figure 5. Example screenshot from the Camelot camera-trap database software, showing the use of the MEWC-encoded meta-data to automatically sort the images by species. This allows for efficient viewing and bulk validation by human experts following the preliminary machine classification (if deemed necessary). In this example the species code (#3) for the Tasmanian Devil is written to the ISO field of the metadata and can be searched in Camelot without any modifications to this software. This system also works well in other popular image-tagging software such as ExifPro[19] or Timelapse[20].



[19] https://www.exifpro.com
[20] https://saul.cpsc.ucalgary.ca/timelapse

Figure 6. Example screenshot from the AddaxAI application, showing the features for deploying MEWC models.



Discussion

As the prevalence of automated sensors in ecology increases, there is a concomitant need to develop efficient ways collate and label the resulting large datasets (Steenweg et al. 2017, Borowiec et al. 2022). For wildlife monitoring, camera traps have become valuable tools for capturing 'big data' on habitat occupancy, population dynamics, and conservation efforts (Hampton et al. 2013, Christin et al. 2019). Although useful AI-driven solutions for automatically processing camera-trap data are now available (e.g., Wildlife Insights: Ahumada et al. 2020), there are many circumstances where third-party offerings, delivered via a universal online portal, is not ideal (Willi et al. 2019, Vélez et al. 2023). Reasons include fees, data-privacy, and limited flexibility (e.g., inability to train custom classifiers that can be tailored to specific wildlife communities). Here we have presented a user-friendly but extensible AI approach (MEWC) that addresses all these concerns. Its open-source pipeline leverages advanced computer-vision techniques for object detection and classification, but it also keeps the details abstracted away for most users. Not only is MEWC easy to train and use with typical camera-trap datasets—as demonstrated in the replicable case study—it also offers a flexibility of deployment on local desktops or cloud-based virtual machines. At its core, MEWC democratises the use of deep learning in wildlife-image processing, by delivering a customisable solution that is still accessible to a broad audience: professional ecologists, practitioners, and citizen scientists.

For anything beyond basic tasks, the capacity to tailor software to specific research aims is a necessity (Nowak et al. 2018, Fordham et al. 2021). With its modular architecture, MEWC is designed such that each component—from image pre-processing to final classification and expert validation—can be plugged in or out, lending itself to high adaptability for a wide range of tasks associated with processing camera-trap imagery. For instance, setting configuration via environment variables enables users to calibrate object detection settings or use progressive fine-tuning to adapt the model to new data to meet specialised needs, all by simply modifying a single text file of settings. Customisation is thus not confined to those proficient in programming: it offers intuitive controls, and its "code-lite" approach bypasses the otherwise dauntingly steep learning curve associated with implementing customised AI-classifier workflows from scratch. Deployment is also straightforward, thanks to Docker containers that encapsulate all dependencies, ensuring it works within nearly any modern computing environment. Moreover, MEWC is fully compatible with the free Camelot database (Hendry and Mann 2018) and AddaxAI application, allowing for a straightforward data-exchange process and user-friendly deployment options (Fig. 6). This interoperability underscores MEWC's role as not just a stand-alone application, but as a part of a broader open-source toolkit.

Cost-effectiveness and scalability are key features of MEWC. While commercial solutions for image analysis offer ease of use, they lack the 'agile' adaptability and affordability central to MEWC's lean design. Indeed, their price structure might be unsustainable for large surveys. For example, for-profit consultancies typically charge 1 cent per image processed (irrespective of animal or blank)[21], 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). Beyond its obvious cost benefits, MEWC does not compromise on technical robustness. Compatible with NVIDIA GPUs, the system allows accelerated model fitting and inference. It also integrates seamlessly with arbitrarily expandable cloud-compute

[21] 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.

platforms. This includes both national high-performance computing infrastructure (like Nectar in Australia, which is free for research projects that meet priority goals, such as environmental sustainability) and commercial offerings for Amazon Web Services, Microsoft Azure, Lambda, etc., which offer scalability while maintaining cost efficiency. When training or inference is run on a cloud provider, we have, by default, configured Terraform and Ansible to build the virtual machine on demand, and tear it down immediately upon job completion. This avoids any unexpected ongoing costs that can be incurred by idle software as infrastructure. Performance metrics further bolster its appeal. While exact figures will depend on specific configurations and ecological foci, MEWC's combination of speed, accuracy, and customisability (see Case Study)—for tasks ranging from fine-tuning classifiers based on curated local species lists (to avoid misclassifications of irrelevant species) or even for 'mining' historical camera-trap data—sets it apart from online portals. This is critical for projects where nuanced detection can significantly inform conservation strategies. If using cloud deployment, the system's architecture permits multiple users within an organisation to deploy it simultaneously without encountering bottlenecks.

Released under a CC-BY 4.0 creative commons license[22], we have made MEWC an open-source project to encourage community-driven enhancements—enabling an evolving, synergistic solution that avoids the rigidity of commercial alternatives. This approach allows for full transparency in project maintenance, release of software updates, and ongoing capability improvements (Fitzgerald 2006). These will be informed by end-user feedback and seek to incorporate new technical developments as they arise, ensuring that MEWC remains at the cutting-edge while maintaining its user-friendly ethos. Both the code and pre-built containerised applications are publicly available for download and customisation, from GitHub for the code base, to Docker Hub for distribution. This architecture ensures that even individuals with no specialised programming skills or knowledge of deep-learning can effectively engage with the system to classify and label their extensive camera-trap datasets. It also allows researchers to retain full control over their data, ameliorating concerns over data privacy and jurisdiction. As such, MEWC represents a pragmatic and holistic solution, merging performance, customisability, and financial accessibility, in one robust package.

Beyond its primary role in multi-species classification tasks, MEWC could also be used to develop further automated data breakdowns, such as within-species categorisation based on visual morphology. For example (Chen et al. 2019, Ferreira et al. 2020, Clapham et al. 2022), computer vision can be effective for automatically identifying an individual's gender (for sexually dimorphic species), diagnosing infection status for diseases with visibly diagnostic symptoms (e.g., alopecia, Devil Facial Tumour Disease), determining colour or pattern morphs for species with distinct markings, or tracking prey use for invasive species (e.g., denning female cats will carry small prey items back to their kittens). All that is required to undertake these tasks with MEWC is to train a classifier with labelled examples of these feature categories, and then the species and state classifiers can be daisy-chained to achieve the desired hierarchical classification. New modules could be added to MEWC to allow the system to alternatively train a multi-output model that does both classifications simultaneously, or for automated ecological analysis on the CSV output files.

There will be cases where no (or few) labelled training data exist for a new classification task. In such cases, MEWC can still bootstrap new-image labelling, using an approach called active learning (Norouzzadeh et al. 2021). For example, a sample of raw camera data can be run through the detect and snip modules (see Fig. 1) to create a pre-formatted batch of snips. These can then be manually

[22] https://creativecommons.org

sorted by bulk viewing them in an image browser and using drag-and-drop to allocate them into labelled folders. If tagged images are available (with the tags written, for instance, to a text file or the image metadata), then a Python or Bash/PowerShell script can be used to pre-sort the images and then detect-and-snip can be used to extract the final training images. Online biodiversity databases like gbif.org, inaturalist.org and ebird.org also contain many suitable labelled images that can be used to kick-start a customised classifier. Thereafter, a preliminary classifier model can be used to classify new data and suggest classifications for another round of snips, allowing for a semi-supervised approach facilitating a rapid expansion of the pool of labelled training data, in an iterative fashion.

The practical demand for a system like MEWC is high (Steenweg et al. 2017, Vélez et al. 2023). From the external end-user perspective, many agencies are seeking to adopt approaches that enhance the cost-effectiveness of their time-sensitive efforts in conservation monitoring (e.g., for threatened species detection, community composition, or turnover across sites in different landscape contexts, or for invasive-species management) (Nowak et al. 2018). Natural Resource Management organisations now routinely use camera traps for responding to suspected incursions of invasive vertebrates, as a follow-up tool to confirm eradication, for site surveillance, and for confirmation of rare species' presence. To make use of wildlife cameras in these monitoring roles, the major bottleneck has been the expert time required for image labelling (Nguyen et al. 2017). An easy-to-deploy AI-based approach like MEWC, which can be custom built for each task, offers the capacity to rapidly extract and automatically analyse ecological information from bulk collections of raw image data. This holds enormous potential rapid responses, as the data-processing step after retrieval of field imagery becomes almost instantaneous, yet robust (Whytock et al. 2021).

Conclusion

The MEWC workflow is a modern tool for ecological and conservation research, offering a comprehensive, customisable, and freely available solution for classifying wildlife-image data. Bridging the latest AI tools in computer vision with an intuitive user experience, its modular design and cloud-computing compatibility invite scalability and adaptability. Beyond immediate applications in species classification, MEWC's architecture is future-proofed, capable of accommodating new analytical modules for nuanced tasks, from within-species breakdowns to disease surveillance. Our commitment to open-source development ensures that MEWC will continue to evolve through community input, promoting highly customisable approaches to working with camera-trap data sets.

Acknowledgements

This work was funded by the Australian Research Council through projects FT160100101 and CE170100015 to BWB. Hardware and virtual machine Infrastructure support was provided through Australian Research Data Commons Nectar Research Cloud, funded by the Australian government and the University of Tasmania's High Performance Computing facility (tpac.org.au).

Code and Data Availability

Code (open source), tutorials and vignettes: http://github.com/zaandahl/mewc

Data (case study): Brook, B.W. & Buettel, J.C. (2023) Mega-Efficient Wildlife Classifier (MEWC) Case Study. https://dx.doi.org/10.25959/wm5g-b990

Author Contributions

BWB wrote the manuscript, collected the primary data, generated the results and co-designed / co-wrote the Python code for MEWC with PPR and RZA. JCB co-conceived the project and collected data. PvL developed AddaxAI and integrated MEWC within this GUI-based tool. RZA lead the Docker code development and GitHub deployment. All co-authors commented on the draft.

Conflict of Interest Disclosure

PvL is the developer of AddaxAI (https://github.com/PetervanLunteren/AddaxAI). However, this project is non-commercial and fully open source.

The authors declare they have no other conflict of interest relating to the content of this article.

References

Ahumada, J. A., E. Fegraus, T. Birch, N. Flores, R. Kays, T. G. O'Brien, J. Palmer, S. Schuttler, J. Y. Zhao, and W. Jetz. 2020. Wildlife insights: A platform to maximize the potential of camera trap and other passive sensor wildlife data for the planet. Environmental Conservation **47**:1-6.

Beery, S., D. Morris, and S. Yang. 2019. Efficient pipeline for camera trap image review. arXiv preprint arXiv:1907.06772.

Boitani, L. 2016. Camera trapping for wildlife research. Pelagic Publishing Ltd.

Borowiec, M. L., R. B. Dikow, P. B. Frandsen, A. McKeeken, G. Valentini, and A. E. White. 2022. Deep learning as a tool for ecology and evolution. Methods in Ecology and Evolution **13**:1640-1660.

Bubnicki, J. W., M. Churski, and D. P. Kuijper. 2016. Trapper: An open source web-based application to manage camera trapping projects. Methods in Ecology and Evolution **7**:1209-1216.

Chen, R., R. Little, L. Mihaylova, R. Delahay, and R. Cox. 2019. Wildlife surveillance using deep learning methods. Ecology and Evolution **9**:9453-9466.

Chollet, F. 2021. Deep learning with Python. Simon and Schuster.

Christin, S., É. Hervet, and N. Lecomte. 2019. Applications for deep learning in ecology. Methods in Ecology and Evolution **10**:1632-1644.

Clapham, M., E. Miller, M. Nguyen, and R. C. Van Horn. 2022. Multispecies facial detection for individual identification of wildlife: a case study across ursids. Mammalian Biology **102**:943-955.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. Pages 248-255 *in* 2009 IEEE conference on computer vision and pattern recognition. Ieee.

Falzon, G., C. Lawson, K.-W. Cheung, K. Vernes, G. A. Ballard, P. J. S. Fleming, A. S. Glen, H. Milne, A. Mather-Zardain, and P. D. Meek. 2020. ClassifyMe: A Field-Scouting Software for the Identification of Wildlife in Camera Trap Images. Animals **10**:58.

Ferreira, A. C., L. R. Silva, F. Renna, H. B. Brandl, J. P. Renoult, D. R. Farine, R. Covas, and C. Doutrelant. 2020. Deep learning-based methods for individual recognition in small birds. Methods in Ecology and Evolution **11**:1072-1085.

Fitzgerald, B. 2006. The transformation of open source software. MIS quarterly:587-598.

Fordham, D. A., S. Haythorne, S. C. Brown, J. C. Buettel, and B. W. Brook. 2021. poems: R package for simulating species' range dynamics using pattern-oriented validation. Methods in Ecology and Evolution **12**:2364-2371.

Greenberg, S., T. Godin, and J. Whittington. 2019a. Design patterns for wildlife-related camera trap image analysis. Ecology and Evolution **9**:13706-13730.

Greenberg, S., T. Godin, and J. Whittington. 2019b. Design patterns for wildlife-related camera trap image analysis. Ecology and Evolution **9**:13706-13730.

Hampton, S. E., C. A. Strasser, J. J. Tewksbury, W. K. Gram, A. E. Budden, A. L. Batcheller, C. S. Duke, and J. H. Porter. 2013. Big data and the future of ecology. Frontiers in Ecology and the Environment **11**:156-162.

Hendry, H., and C. Mann. 2018. Camelot—intuitive software for camera-trap data management. Oryx **52**:15.

Huh, M., P. Agrawal, and A. A. Efros. 2016. What makes ImageNet good for transfer learning? arXiv preprint arXiv:1608.08614.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**.

LeCun, Y., and Y. Bengio. 1995. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**:1995.

Miell, I., and A. Sayers. 2019. Docker in practice. Simon and Schuster.

Nguyen, H., S. J. Maclagan, T. D. Nguyen, T. Nguyen, P. Flemons, K. Andrews, E. G. Ritchie, and D. Phung. 2017. Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring. Pages 40-49 *in* 2017 IEEE international conference on data science and advanced Analytics (DSAA). IEEE.

Norouzzadeh, M. S., D. Morris, S. Beery, N. Joshi, N. Jojic, and J. Clune. 2021. A deep active learning system for species identification and counting in camera trap images. Methods in Ecology and Evolution **12**:150-161.

Nowak, J. J., P. M. Lukacs, M. A. Hurley, A. J. Lindbloom, K. A. Robling, J. A. Gude, and H. Robinson. 2018. Customized software to streamline routine analyses for wildlife management. Wildlife Society Bulletin **42**:144-149.

Penn, M. J., V. Miles, K. L. Astley, C. Ham, R. Woodroffe, M. Rowcliffe, and C. A. Donnelly. 2024. Sherlock-A flexible, low-resource tool for processing camera-trapping images. Methods in Ecology and Evolution **15**:91-102.

Pollock, K. H., J. D. Nichols, T. R. Simons, G. L. Farnsworth, L. L. Bailey, and J. R. Sauer. 2002. Large scale wildlife monitoring studies: statistical methods for design and analysis. Environmetrics: The official journal of the International Environmetrics Society **13**:105-119.

Santos, C. F. G. D., and J. P. Papa. 2022. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. ACM Computing Surveys (CSUR) **54**:1-25.

Schneider, S., S. Greenberg, G. W. Taylor, and S. C. Kremer. 2020. Three critical factors affecting automated image species recognition performance for camera traps. Ecology and Evolution **10**:3503-3517.

Steenweg, R., M. Hebblewhite, R. Kays, J. Ahumada, J. T. Fisher, C. Burton, S. E. Townsend, C. Carbone, J. M. Rowcliffe, and J. Whittington. 2017. Scaling-up camera traps: Monitoring the planet's biodiversity with networks of remote sensors. Frontiers in Ecology and the Environment **15**:26-34.

Tabak, M. A., M. S. Norouzzadeh, D. W. Wolfson, S. J. Sweeney, K. C. VerCauteren, N. P. Snow, J. M. Halseth, P. A. Di Salvo, J. S. Lewis, and M. D. White. 2019. Machine learning to classify animal species in camera trap images: Applications in ecology. Methods in Ecology and Evolution **10**:585-590.

van Lunteren, P. 2023. EcoAssist: A no-code platform to train and deploy custom YOLOv5 object detection models. The Journal of Open Source Software **8**:5581.

Vélez, J., W. McShea, H. Shamon, P. J. Castiblanco-Camacho, M. A. Tabak, C. Chalmers, P. Fergus, and J. Fieberg. 2023. An evaluation of platforms for processing camera-trap data using artificial intelligence. Methods in Ecology and Evolution **14**:459-477.

Whytock, R. C., J. Świeżewski, J. A. Zwerts, T. Bara-Słupski, A. F. Koumba Pambo, M. Rogala, L. Bahaa-el-din, K. Boekee, S. Brittain, and A. W. Cardoso. 2021. Robust ecological analysis of camera trap data labelled by a machine learning model. Methods in Ecology and Evolution **12**:1080-1092.

Willi, M., R. T. Pitman, A. W. Cardoso, C. Locke, A. Swanson, A. Boyer, M. Veldthuis, and L. Fortson. 2019. Identifying animal species in camera trap images using deep learning and citizen science. Methods in Ecology and Evolution **10**:80-91.

Xu, M., S. Yoon, A. Fuentes, and D. S. Park. 2023. A comprehensive survey of image augmentation techniques for deep learning. Pattern Recognition:109347.

Young, S., J. Rode-Margono, and R. Amin. 2018. Software to facilitate and streamline camera trap data management: A review. Ecology and Evolution **8**:9947-9957.