PlanktonFlow : hands-on deep-learning classification of plankton images for biologists

Hugo Walter[1], Caroline Gorzerino[1], Marc Collinet[1], Béatrice Porcon[1], François Martignac[1], and Eric Edeline[1]

DOI not yet assigned

Abstract

High-resolution biodiversity monitoring is crucial to understand the complex dynamics of ecosystems. High throughput image-acquisition devices now provide biologists with a wealth of data, but mainstream, automated image-classification workflows are still lagging behind, while high-performance classification models remain hardly manageable by most biologists.

To fill this gap, we developed PlanktonFlow, an end-to-end, fully automated deep-learning pipeline for plankton imagery intended for biologists having minimal technical expertise with machine learning. PlanktonFlow automates image pre-processing, model training and calculation of performance metrics for four different convolution neural network architectures (ResNet, DenseNet, EfficientNet, and YOLOv11), as well as inference on novel datasets. We highlight the potentialities of PlanktonFlow by comparing the relative performances of the four models and of the web-service EcoTaxa on classifying images from freshwater samples including 76 classes of both living and non-living objects.

EfficientNet-B5 achieved the highest macro-averaged F1 score, outperforming EcoTaxa as well as other models. Despite class imbalance and morphological complexity, all models generalized well across frequent and rare taxa, highlighting the relevancy of the pre-processing routine. Error analyses stressed that classification challenges arose from high intra-class variability and high inter-class similarity.

To ease an appropriation and further developments by the community, PlanktonFlow is open source and has a modular structure. We foresee that future work could leverage semi-supervised learning, test the pipeline on other imaging systems, and expand its applicability to other biological systems.

Keywords: Computer vision, Image classification, Python, Plankton Classification.

[1]DECOD (Ecosystem Dynamics and Sustainability), INRAE, IFREMER, Institut Agro, Rennes, France.

Correspondence
Eric.Edeline@inrae.fr

Introduction

High-throughput biodiversity monitoring, as enabled by modern automatic sensors, provides unprecedented capacity to track rich, multidimensional data streams and, ultimately, to accurately forecast ecological dynamics across broad spatial and temporal scales (Besson et al., 2022, Weinstein, 2018). However, the volume and heterogeneity of these data often exceed the routine processing capacity of traditional analysis workflows, and thus create bottlenecks that hinder scientific progress.

This tension is particularly clear in aquatic systems, where high-frequency plankton monitoring has advanced rapidly over the last two decades. A diversity of imaging-in-flow and camera systems now cover a wide spectrum of plankton sizes and taxonomic diversity. Submersible, high-frequency instruments like the Imaging FlowCytobot (Olson and Sosik, 2007) or the Underwater Vision Profiler (UVP, Picheral et al., 2010) operate autonomously *in situ* to capture continuous records of plankton. In parallel, benchtop imaging flow cytometers for plankton, such as the FlowCAM® (Sieracki et al., 1998), are widely used to analyse water samples *ex-situ* in the laboratory or under shipboard settings.

In practice, extracting taxonomic information from the wealth of images produced by these machines still remains a major challenge. Softwares shipped with commercial instruments (for example, FlowCAM's VisualSpreadsheet) typically require substantial manual curation to reach reliable taxonomic automation and make it impossible to process large datasets (Owen et al., 2022). To scale up classification, the community has adopted a web-based platform, EcoTaxa (Irisson et al., 2022; Picheral et al., 2025), which provides a user-friendly collaborative graphical user interface and assists taxonomic labelling using fast machine-learning algorithms. EcoTaxa is a great tool when projects are small enough to allow a validation of all images by experts. However, EcoTaxa unfortunately lacks tools to easily measure classification performances (accuracy, recall, confusion matrix...), making it less relevant for very large projects where model predictions can not be validated by eye. Additionally, as a web service, EcoTaxa must trade speed and energy efficiency against prediction performance, and thus does not implement the best-predicting, but slow and energy-hungry models.

At the same time, recent studies show on various benchmark plankton datasets that modern deep learning approaches (CNNs, Vision Transformers) can achieve classification accuracies that go far beyond what can be achieved using EcoTaxa (Eerola et al., 2024). However, several obstacles prevent the routine adoption of these novel approaches by biologists: variation among datasets often preclude a direct transfer of methods among instruments and environments, severe class imbalance and rarity of many taxa impose custom-trained models, building high-quality training datasets requires extensive expert time for manual annotation and validation, and the computational/engineering overhead required to train, deploy and maintain robust models in operational settings are often beyond-of-reach for non-specialists. Consequently, despite their high classification performances, state-of-the-art models remain excluded from laboratory workflows.

These factors create a clear gap: many biologists are in need of modern deep-learning methods but are unable to implement them. This community therefore needs a new, hands-on, open and reproducible pipeline that (i) ingests instrument outputs (FlowCAM, IFCB, UVP, ...), (ii) comes close to state of the art performances, and (iii) ships with ready-to-use, well-documented and

easy scripts to train modern deep-learning classifiers. PlanktonFlow aims to addresses these needs.

Specifically, with PlanktonFlow, biologists having no deep-learning expertise can, using just a few single-line Python commands, obtain high-quality plankton classifications from raw segmented images. The package includes pretrained models ready for immediate use, as well as streamlined workflows to automatically train a custom model on new datasets. Additionally, for those who wish to refine or extend the approach, PlanktonFlow allows hyperparameter tuning on the new dataset, as well as transparent, reproducible training recipes, clear step-by-step instructions, and all the configuration files needed to adapt the pipeline to other instruments or taxa. In this paper, we demonstrate the usage of PlanktonFlow on a custom plankton dataset from our own freshwater mesocosm experiments.

Material and methods

Dataset

Raw dataset. The plankton image dataset used in this study was acquired from monthly water samples taken in 12 outdoor, fishless mesocosms in 2022 and 2023 (24 months). Mesocosms were installed in 2020 at the PEARL, the INRAE experimental platform for aquatic ecology and ecotoxicology (U3E) located in Rennes, France (Platform PEARL). The mesocosms were initially seeded with pond sediments, such that a rich vegetal and animal community rapidly developed. Organisms were also free to colonize mesocosms by aerial dispersal. In 2021, red swamp crayfish (*Procambarus clarkii*) were introduced in eight mesocosms and eradicated all the vegetation. Hence, our image dataset contained diverse plankton taxa from both vegetated and non-vegetated natural freshwater ecosystems.

Imaging was performed using the FlowCAM® (Fluid Imaging Technologies Inc.), a high-throughput imaging cytometer designed for particle analysis in liquid media. Two separate FlowCAM instruments were used to optimize imaging for different organismal size ranges: a FlowCAM 8110 dedicated to microplankton (50 to 200 μm) and a FlowCAM Macro dedicated to macroplankton (0.2 to 2mm). Following FlowCAM acquisition, raw images in .tiff format were segmented using the Zooprocess ImageJ macro (Jalabert et al., 2025) to obtain low-resolution images (approximately 400 by 400 px), each containing one object.

These objects exhibited a wide spectrum of visual complexity, including variations in pose, orientation, occlusion, transparency, and morphological deformation. This heterogeneity reflected the natural complexity of water samples including target planktonic organisms, non-target biological material (e.g., terrestrial organisms), and detritus and other particles.

Segmented images were uploaded to EcoTaxa for manual construction of our dataset of labelled images. In building the dataset, we retained a large number of classes reflecting the diversity of objects effectively encountered, rather than reducing the label space to ease subsequent automatic classification. We also chose not to check the dataset for classification errors (a highly time-consuming task), because our goal was not to establish a new "benchmark" reference model with the highest classification performances, but rather to develop an operational pipeline. Therefore, our dataset is not a reference dataset like is, for instance, WHOI-Plankton (Orenstein et al., 2015).

Data pre-processing. To ensure the reliability and statistical relevancy of the training data, we applied a pruning strategy that removed extremely rare classes. Indeed, the dataset exhibited a strongly skewed class distribution, which was characteristic of ecological samples: the top 33 most abundant taxa accounted for over 70% of all annotated images, while the remaining 73 taxa collectively contributed the remaining 30%. Specifically, any class with fewer than 100 manually-labelled images (i.e., less than 0.4 individual per mesocosm per month in our experimental design) was excluded from the final dataset. This decision was motivated by two factors: (1) very low sample sizes tend to introduce high-variance gradients that hinder stable model convergence, and (2) such elusive classes often will not be used in later intended research. After pruning, the number of valid classes was reduced from $N_{original}$=106 to N_{final}=76, with the removed classes contributing marginally to the overall image volume.

A critical preprocessing step in the pipeline involved detecting and removing technical image artifacts, particularly scale bars that were embedded in some of the original imagery. Inconsistency in the presence of scale bars was due to varying settings in the different versions of Zooprocess. To remove all scale bars, we employed a dedicated object-detection model based on the YOLOv8 architecture, which we fine-tuned specifically to detect and localize scale bars. Because scale bars are visually distinctive and consistently positioned, the task was relatively straightforward for a modern model like YOLO. As a result, only about 1,000 hand-labelled images were needed to train a model that achieved a close-to-100 % accuracy.

Data Augmentation. The performances of deep learning models often deteriorate when exposed to imbalanced training distributions, especially in multi-class settings with sparse and heterogeneous categories. In the context of our plankton classification task, pronounced class imbalance was observed across taxa, primarily due to natural ecological abundance gradients. To mitigate this issue and to enhance the model's generalization capability, we implemented a *class-aware data augmentation strategy* aimed at enriching under-represented categories in the training set.

Specifically, we augmented data using the `Albumentations` library, where each transformation is applied independently with a given probability. The transformations included horizontal flipping (50%), vertical flipping (20%), 90-degree rotation (30%), brightness and contrast adjustment (40%), and hue/saturation modulation (30%). Since these operations are sampled separately, multiple augmentations may be applied to the same image.

To standardize the training distribution, we defined an *augmentation threshold* of 1,500 images per class. For any class below this threshold, we calculated the required number of synthetic samples and distributed the augmentation load across the available original images using a capped multiplier (maximum 10 augmentations per original). This process ensured a controlled and balanced enrichment of the dataset while maintaining fidelity to the original morphological patterns.

Finally, to ensure robust model evaluation, we split the final dataset into three disjoint subsets:

- $\mathcal{D}_{\text{train}}$: Training Set (70%): 159,687 images
- $\mathcal{D}_{\text{val}}$: Validation Set (20%): 25,921 images
- $\mathcal{D}_{\text{test}}$: Test Set (10%): 12,961 images

The split was performed in a stratified manner across the 76 classes to preserve class distributions, ensuring that rare classes remained proportionally represented across all subsets.

Baseline & model selection

We considered EcoTaxa as our baseline (see Supplementary Information, Walter et al., 2025), and compared its performances against four commonly-used deep-learning architectures : ResNet (He et al., 2016), DenseNet(Huang et al., 2018), EfficientNet (Tan and Le, 2020) and YOLOv11 (Khanam and Hussain, 2024). All four models were fine-tuned by their developers using transfer learning, initialized with ImageNet-pretrained weights, as prior work shows that out-of-domain initialization can outperform in-domain pretraining on plankton classification: Maracani et al., 2023 demonstrated that ResNet-50 models initialized on ImageNet-22K achieved $\sim 6\%$ higher accuracy than models pretrained on large plankton datasets..

We chose ResNet in our candidate-model set for its use of residual connections, which facilitates gradient flow and enables the training of deep networks without degradation. Variants like ResNet-50 offer a solid balance between accuracy and resource use, making them a common baseline for transfer learning. DenseNet connects each layer to every other layer in a feed-forward manner, improving gradient propagation and promoting feature reuse. This architecture achieves strong generalization with fewer parameters, making it well-suited for complex but data-limited tasks. EfficientNet applies compound scaling to jointly optimize network depth, width, and input resolution, achieving high accuracy with fewer parameters and FLOPs (Floating Point Operations, measure computational complexity). Its scalable variants (B0–B7) offer flexibility between speed and performance, with smaller models enabling fast prototyping. Finally, the YOLOv11 classifier, adapted from a real-time object detection architecture, was included to benchmark a modern, user-friendly deep learning workflow. It was trained end-to-end using the Ultralytics framework with default settings, demonstrating the accessibility of preconfigured pipelines for non-expert users.

Prior to comparing the respective classification performances of the models, we optimized ResNet, DenseNet and EfficientNet using loss functions and hyperparameter tuning (see below). We did not optimize YOLOv11 as we chose this architecture to demonstrate the performances of out-of-the-box deep-learning solutions. To comprehensively assess classification performances, we employed multiple standard metrics: accuracy, precision, recall, and F1-score (described in details below). These metrics were calculated in both macro-averaged and weighted-averaged forms to provide insight into both overall and per-class performance. These metrics were computed using the `scikit-learn` library, a standard in machine learning.

Accuracy. measures the proportion of correctly classified samples over all samples. In this work, we report both top-1 accuracy (Acc@1) and top-5 accuracy (Acc@5):

$$\text{Accuracy@k} = \frac{1}{N}\sum_{i=1}^{N} 1(y_i \in \widehat{Y}_i^{(k)}), \tag{1}$$

where N is the total number of samples, y_i is the ground truth label, and $\widehat{Y}_i^{(k)}$ is the set of the model's top-k predicted class labels for sample i. When $k = 1$, this reduces to standard (top-1) accuracy. 1 denotes the indicator function.

Precision. for a class c is defined as the ratio of true positives (TP) to the sum of true positives and false positives (FP):

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}. \tag{2}$$

In the multiclass setting, TP_c denotes the number of samples correctly predicted as belonging to class c. FP_c counts the samples incorrectly predicted as class c.
Recall. (also called sensitivity) for class c measures the ratio of true positives to the sum of true positives and false negatives FN_c, which correspond to samples of class c that were mistakenly classified as another class:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}. \tag{3}$$

F1-score. for class c is the harmonic mean of precision and recall:

$$\text{F1}_c = 2 \times \frac{\text{Precision}_c \times \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}. \tag{4}$$

Macro-averaged metrics. compute the unweighted mean across classes:

$$\text{Metric}_{\text{macro}} = \frac{1}{C} \sum_{c=1}^{C} \text{Metric}_c, \tag{5}$$

where C is the number of classes.
Weighted-averaged metrics. weight each class metric by its support (number of true samples in that class):

$$\text{Metric}_{\text{weighted}} = \frac{1}{N} \sum_{c=1}^{C} n_c \times \text{Metric}_c, \tag{6}$$

where n_c is the number of samples in class c.

The combination of accuracy, precision, recall, and F1-score provides a multi-faceted evaluation of model performance. Accuracy summarizes overall correctness but can be misleading in imbalanced datasets. Precision and recall highlight the model's trade-off between false positives and false negatives. F1-score balances these two aspects, which is crucial when classes are unevenly represented.

For model selection, the primary metric we used was the macro-averaged F1-score (F1_{macro}). This metric equally weights the F1-score of each class, preventing dominant classes from skewing the evaluation and ensuring robust performance across all categories.

Model Optimization Strategies

Why simple fine-tuning is not sufficient. Plankton images pose challenges for automated classification that make standard fine-tuning insufficient. Fine-tuning involves adapting a model pretrained on a large generic dataset (e.g., ImageNet) to a new task, leveraging general features but not always capturing the specific variability of plankton. Some classes are rare, others are morphologically similar, and individuals within a class can vary greatly in size, shape, and orientation. Standard fine-tuning tends to favour majority classes and may fail to generalize to rare or difficult examples. To address this problem, we applied additional optimization strategies, including specialized loss functions to handle class imbalance and challenging samples, as well as hyperparameter tuning to identify optimal learning configurations.

Loss Functions. A loss function quantifies how well the model's predictions match the true labels. Conceptually, it can be thought of as a supervisory signal: the model attempts to minimize this loss to improve its predictions. Different loss functions emphasize different aspects of learning and can be critical in datasets characterized by class imbalance or high intra-class variability.

A common baseline is the standard cross-entropy loss $\mathcal{L}_{\text{CE}}$ (Equation 7). It treats each class equally and penalizes incorrect predictions proportionally to the predicted probability assigned to the true class:

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{C} y_c \log(\widehat{y}_c). \tag{7}$$

Here, y_c represents the ground-truth one-hot label for class c, and $\widehat{y}_c$ is the model's predicted probability for that class. While effective in balanced datasets, this formulation tends to underperform in the presence of rare classes, which are given equal treatment to frequent classes despite their limited representation.

Thus, to correct this, the weighted cross-entropy loss $\mathcal{L}_{\text{WCE}}$ (Equation 8) introduces class-specific weights w_c to counteract this imbalance :

$$\mathcal{L}_{\text{WCE}} = -\sum_{c=1}^{C} w_c \cdot y_c \log(\widehat{y}_c). \tag{8}$$

In this approach, weights are typically set inversely proportional to class frequency, effectively amplifying the contribution of rare classes during training. This ensures that the model allocates sufficient learning capacity to under-represented taxa, reducing bias toward the dominant classes.

Another approach is to use Focal Loss $\mathcal{L}_{\text{focal}}$ (Equation 9) (Lin et al., 2018) which further improves model performance by emphasizing difficult examples :

$$\mathcal{L}_{\text{focal}} = -\sum_{c=1}^{C} \alpha_c (1-\widehat{y}_c)^{\gamma} \cdot y_c \log(\widehat{y}_c). \tag{9}$$

The modulating factor $(1-\widehat{y}_c)^{\gamma}$ dynamically reduces the loss contribution of well-classified "easy" examples and focuses learning on misclassified or rare instances. Here, γ is a focusing parameter that controls the degree of emphasis on difficult cases, while α_c allows optional per-class weighting. This loss function is particularly useful when both class imbalance and intra-class variability are present, as it directs learning towards the most challenging examples.

Finally, Label Smoothing (Szegedy et al., 2015) (Equations 10 and 11) is a regularization technique that reduces overconfidence in predictions and encourages better generalization :

$$\tilde{y}_c = \begin{cases} 1-\epsilon & \text{if } c = y \\ \frac{\epsilon}{C-1} & \text{otherwise.} \end{cases} \tag{10}$$

The corresponding cross-entropy loss with label smoothing becomes:

$$\mathcal{L}_{\text{LS}} = -\sum_{c=1}^{C} \tilde{y}_c \log(\widehat{y}_c). \tag{11}$$

In Equation 10, the smoothing parameter ϵ determines the fraction of probability mass redistributed to other classes. By avoiding assigning full probability mass to a single class, label smoothing mitigates the tendency of the model to make overconfident predictions, which is

particularly important when some classes are visually similar. For implementation details on considered loss functions described below, see Supplementary Information (Walter et al., 2025).

Hyperparameters: the knobs that control learning. Hyperparameters are parameters that define the configuration of the learning process but are not directly learned from the data. They play a critical role in determining how effectively a model can be trained and how well it generalizes to unseen data. Conceptually, hyperparameters can be thought of as 'knobs' that adjust how the model learns, analogous to changing the focus when examining a specimen under a microscope.

Due to the large number of hyperparameters and their complex interactions, manually selecting an optimal configuration is challenging. To address this, we employed Bayesian optimization via Weights & Biases Sweeps (Biases, 2023), an automated approach that efficiently explores the hyperparameter space and identifies the settings that maximize the macro-averaged F1 score. Bayesian optimization was chosen over traditional grid or random search methods due to its efficiency in navigating large hyperparameter spaces with fewer training runs, a critical factor given the computational constraints.

Key hyperparameters include :

- Learning rate: the step size used to update the model's parameters at each iteration, affecting convergence speed and stability.
- Batch size: the number of images processed before updating the model parameters, which influences the variance of gradient estimates and memory usage.
- Optimizer type: the algorithm used to perform parameter updates, such as Adam of AdamW, which differ in how the model adjusts weights.
- Weight decay: a regularization term that discourages overly large parameter values, helping to prevent overfitting.

Models were trained for a fixed number of epochs and batch size, set to 30 and 64, respectively. An epoch corresponds to a full pass of the training dataset through the network, and 30 epochs were found sufficient to achieve convergence in performance while minimizing the risk of overfitting. The batch size refers to the number of training samples processed simultaneously before updating the model weights. A batch size of 64 was chosen to optimize the throughput of our hardware setup, ensuring efficient utilization of computational resources without compromising training stability. Details of the learning rate scheduler and early stopping strategy are provided in the Supplementary Information (Walter et al., 2025).

Table 1 provides a detailed overview of the hyperparameters considered during tuning, while Table 2 lists the model variants and their approximate number of parameters. This systematic approach ensures that each architecture is trained under conditions optimized for handling our specific classification problem.

Hardware and Software Environment

All experiments were conducted on a workstation running Ubuntu 24.04.2 LTS (Noble Numbat). The system was equipped with an NVIDIA Quadro RTX 6000 GPU featuring 24 GB of VRAM. The GPU was utilized at nearly full capacity during training, as indicated by the `nvidia-smi` output, with over 20 GB of memory used by the training process. The driver version used was 570.133.07, along with CUDA 12.8.

Parameter	Type	Values / Range
model_name	Categorical	{"efficientnet", "densenet", "resnet"}
model_variant	Categorical	*Model variants (see Table 2)*
loss_type	Categorical	{"weighted", "focal", "labelsmoothing"}
focal_alpha	Continuous	$[0.1, 0.5]$
focal_gamma	Continuous	$[1.0, 3.0]$
use_per_class_alpha	Boolean	{true, false}
labelsmoothing_epsilon	Continuous	$[0.05, 0.2]$
optimizer	Categorical	{"adamw", "adam"}
learning_rate	Discrete	$\{1 \times 10^{-7}, 5 \times 10^{-7}, 1 \times 10^{-6}, \ldots, 2 \times 10^{-1}\}$
weight_decay	Discrete	$\{1 \times 10^{-7}, 5 \times 10^{-7}, 1 \times 10^{-6}, \ldots, 2 \times 10^{-1}\}$

Table 1 – Hyperparameter search space explored during tuning

Model	Variant	Approx. # Parameters
EfficientNet	b2	9 million
	b3	12 million
	b4	19 million
	b5	30 million
ResNet	18	11.7 million
	34	21.8 million
	50	25.6 million
	101	44.5 million
	152	60.2 million
DenseNet	121	8 million
	161	28.7 million
	169	14.1 million
	201	20 million

Table 2 – Details of model variants and their approximate number of parameters

The training environment was managed via `python3` and relied heavily on GPU acceleration for faster convergence of deep learning models. The use of a professional-grade GPU ensured minimal training time, which is crucial when performing hyperparameter optimization and model evaluation across multiple configurations.

Results

Best-model hyperparameters

The best-performing configuration for each architecture was identified through systematic hyperparameter optimization (see Methods Hyperparameters). Table 3 reports the final settings retained for ResNet-152, DenseNet-161, and EfficientNet-B5. Although the three models converged toward broadly similar training regimes, subtle but meaningful differences emerged in the optimal optimization strategy. In particular, ResNet and DenseNet achieved their best performance when trained with the AdamW optimizer, whereas EfficientNet performed best with the classical Adam variant, consistent with prior reports that adaptive optimizers interact differently with depth and width scaling (Dogo et al., 2022). Likewise, EfficientNet favored a slightly larger label-smoothing parameter ($\varepsilon = 0.12$) compared to ResNet ($\varepsilon = 0.05$) and DenseNet

($\varepsilon = 0.06$), suggesting that additional regularization improved its generalization across diverse classes. The optimal learning rates and weight decay values also varied by architecture, highlighting the importance of architecture-specific tuning rather than relying on a single standardized recipe.

Table 3 – Hyperparameters for the best run of each model architecture.

Model	Variant	Optimizer	Loss	Learning Rate	Batch Size	Epochs	Weight Decay
ResNet	ResNet-152	AdamW	Label Smoothing (ε=0.05)	1e-6	64	30	5e-7
DenseNet	DenseNet-161	AdamW	Label Smoothing (ε=0.06)	1e-6	64	30	1e-3
EfficientNet	EfficientNet-B5	Adam	Label Smoothing (ε=0.12)	1e-5	64	30	1e-4

Relative model performances and comparison with EcoTaxa

Table 4 presents the evaluation metrics on the held-out test set for each selected model alongside EcoTaxa.

Table 4 – Performance metrics on the test set for all models. Metrics include weighted and macro-averaged recall (R_w, R_m), precision (P_w, P_m), F1 score ($F1_w$, $F1_m$), and top-1 and top-5 accuracy ($Acc@1$, $Acc@5$). For definitions of the metrics, see Section Baseline & model selection. NB.: $Acc@5$ is not provided for Ecotaxa, which provides only one predicted class per image

Model	R_w	R_m	P_w	P_m	$F1_w$	$F1_m$	Acc@1	Acc@5
ResNet	86.15%	84.74%	85.76%	84.39%	85.77%	84.33%	86.53%	98.88%
DenseNet	86.59%	84.72%	86.24%	84.89%	86.27%	84.58%	86.40%	98.97%
EfficientNet	88.07%	86.25%	87.76%	87.16%	87.69%	**86.20%**	87.90%	98.47%
YOLO	83.35%	82.62%	82.82%	80.49%	82.42%	80.67%	83.33%	98.43%
EcoTaxa	81.91%	80.31%	81.89%	79.78%	81.11%	79.12%	81.91%	NA

ResNet, DenseNet and EfficientNet, the three models that were optimized on our dataset, stood out from the crowd. EfficientNet demonstrated the best overall performance across all metrics. In particular, EfficientNet achieved the highest macro-averaged F1 score, suggesting a high ability to generalize across both frequent and rare classes. While EcoTaxa offered a strong traditional baseline, it lagged behind deep learning models, particularly-so in class-balanced metrics. Notably, it was surpassed by EfficientNet by a margin of +7.08 percentage points in $F1_{macro}$, highlighting the high potential of optimized deep learning models to automate plankton classification. To see F1-Score for all classes for EfficientNet, see Supplementary Information (Walter et al., 2025).

Per-class analysis

While overall performance metrics provide a useful high-level summary, they often obscure important class-level behaviors—particularly in imbalanced datasets like ours. This section investigates prediction dynamics at the class level to uncover specific strengths and weaknesses of the evaluated models.

Class Support vs. F1 Score. To investigate whether class frequency impacted model performance, we used linear regressions of per-class F1 scores against their corresponding support. As shown in Figure 1, regression lines were nearly flat for all models, and adjusted R^2 were low, indicating almost no explanatory power. This shows that class frequency alone was not a strong predictor

of classification quality, and suggests that the procedure was efficient at alleviating the classification problems that arise from class imbalance.

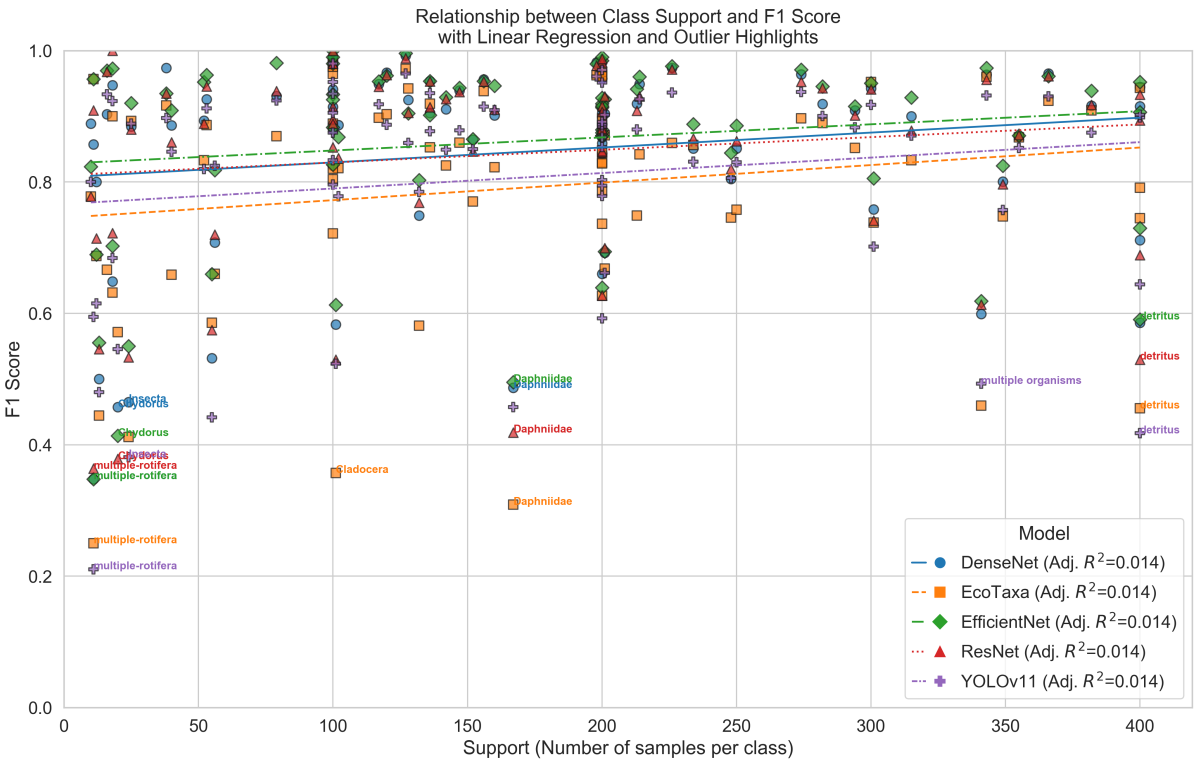
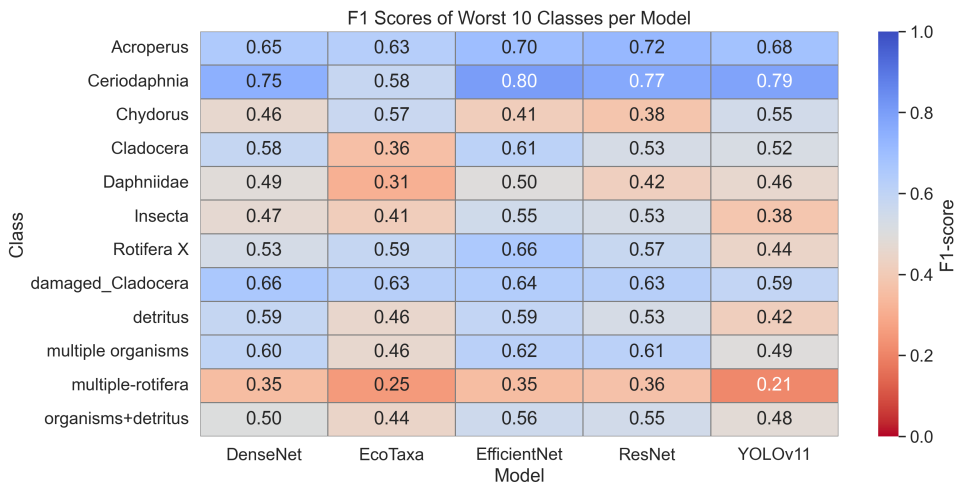


Figure 1 – Scatter plot of per-class F1 score versus class support (log scale) for each model. Rare classes generally perform worse, but some achieve strong F1 scores.

Worst Performing Classes. To better understand model weaknesses, we analyzed the ten lowest-performing classes per model in terms of F1 score. Poor performance was often observed in classes with high intra-class variability like "detritus", "multiple organisms", "Insecta" (Figure 2). While the specific failing classes differ across models, some categories appear recurrently among the bottom ten, suggesting inherent classification difficulty.



Class	DenseNet	EcoTaxa	EfficientNet	ResNet	YOLOv11
Acroperus	0.65	0.63	0.70	0.72	0.68
Ceriodaphnia	0.75	0.58	0.80	0.77	0.79
Chydorus	0.46	0.57	0.41	0.38	0.55
Cladocera	0.58	0.36	0.61	0.53	0.52
Daphniidae	0.49	0.31	0.50	0.42	0.46
Insecta	0.47	0.41	0.55	0.53	0.38
Rotifera X	0.53	0.59	0.66	0.57	0.44
damaged_Cladocera	0.66	0.63	0.64	0.63	0.59
detritus	0.59	0.46	0.59	0.53	0.42
multiple organisms	0.60	0.46	0.62	0.61	0.49
multiple-rotifera	0.35	0.25	0.35	0.36	0.21
organisms+detritus	0.50	0.44	0.56	0.55	0.48

Figure 2 – Per-model F1 scores for the 10 worst-performing classes.

Top confused class pairs. To uncover systematic sources of misclassification, we identified the five most confused class pairs for each model by examining off-diagonal elements in the confusion matrices (Figure 3). Again, we observed that top most confused class pairs are classes with high intra-class variability, with the leading pair being "detritus-dark" confused for "detritus". We *a posteriori* found that confusion among the "Chydoridae" and "Ostracoda" classes were due mainly to human errors in the dataset.

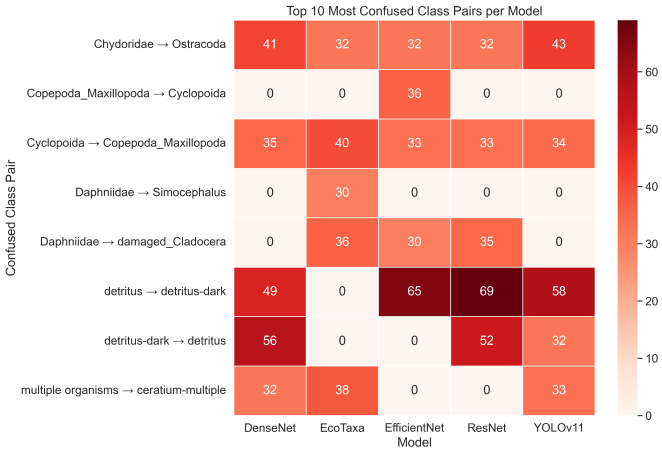


Figure 3 – Top 5 most confused class pairs per model based on confusion matrix analysis.

Error distribution among difficult classes (EfficientNet). Given that EfficientNet was the best-performing model overall, we performed a fine-grained error analysis focused on this architecture. Figure 4 presents the 20 classes with the highest total number of errors, computed as the sum of false positives and false negatives. Interestingly, the distribution of error types across these classes appears relatively balanced, with false positives and false negatives contributing in roughly equal proportions. This suggests that EfficientNet does not exhibit a strong bias toward over- or under-predicting specific classes among its most error-prone predictions, but rather struggles equally with both types of misclassification in these challenging categories.

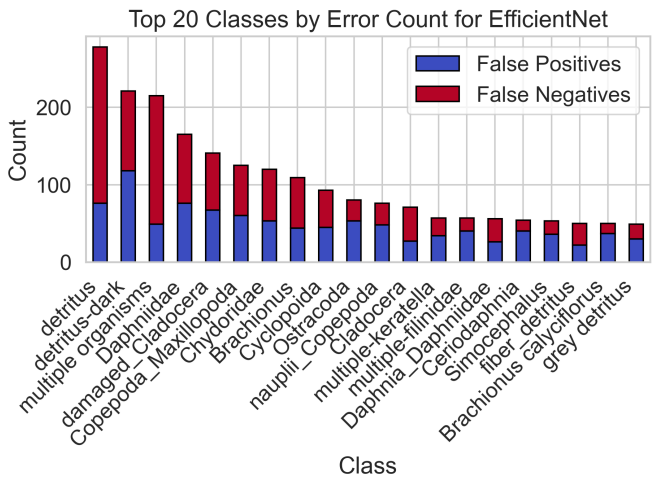


Figure 4 – Top 20 classes by total error count for EfficientNet, with separate bars for false positives and false negatives.

Discussion

This work illustrates the power of a deep learning-based pipeline for taxonomic classification of plankton images, in terms of both classification performance and production of performance metrics. Among the evaluated architectures, the best overall performance was achieved by EfficientNet-B5, while both ResNet and DenseNet achieved comparable results within a 1-2% margin. In contrast, the architectures that we did not optimize (YOLO and EcoTaxa) were clearly lagging behind, illustrating that hyper-parameter optimization is needed to achieve the best model performances.

Importantly, to compare model performances we used a real-world dataset including the "fuzzy" classes (diverse detritus classes, multiple objects on same image...) that are typical of real-world datasets, but that are often dropped from benchmark datasets. Additionally, ou dataset was built from a single round of expert labelling, as is often the case in biological studies due to personnel costs. Hence, our dataset reflected the full variability and noise present under real operational conditions, thus making our results more likely to be relevant for biologists.

As revealed by the per-class analysis, some taxa are inherently hard to classify. This challenge results from high intra-class variability, where individuals within the same class exhibit significant morphological differences, and from low inter-class variability (visually-similar taxa). Sometimes, confusion is made by experts. For instance, in our dataset experts often confused chydoridae for ostracoda, resulting also in confusion by the model. After dataset cleaning, the model no longer confused these two taxa (results not shown), suggesting that the performances of an optimized deep neural network at discriminating plankton classes can match human performances.

We now switch to demonstrating how to use PlanktonFlow to reproduce the above analyses on your own datasets.

PlanktonFlow : a tutorial

The whole pipeline is downloadable from GitHub (PlanktonFlow) and comes with a full-fledged documentation.

Typical workflow with the pipeline. PlanktonFlow is entirely configuration-driven using dedicated YAML configuration files. Users are expected to adapt these files to their own dataset and experimental design. This ensures full reproducibility while keeping commands themselves simple. A typical workflow involves a sequence of preprocessing, training, and inference. Each of these three operations has its own YAML configuration file and may be performed independently.

Data preprocessing. PlanktonFlow accommodates three common data formats:

(1) **Hierarchical dataset**: classical folder-based structure with one folder per class.
(2) **CSV/TSV mapping**: a folder of images with an accompanying CSV/TSV mapping image names to labels.
(3) **EcoTaxa export**: parses EcoTaxa TSV files to extract class labels and associate them with images.

Users can customize preprocessing through configuration files to:

- Prune classes based on a minimum and maximum number of images.
- Perform data augmentation to balance the dataset.
- Optionally remove scale bars using a pre-trained YOLOv8 model.
- Automatically split data into training, validation, and test sets.

At the end of this step, your dataset should be ready to train a model with.

Training. Training is fully configurable via YAML files. Users can select models, variants, loss functions (e.g., focal loss, label smoothing, weighted cross-entropy), optimizers etc. Metrics are tracked in real-time, either locally or online through Weights & Biases. Pre-configured hyperparameters have been validated on our plankton dataset, allowing users to achieve good performance out-of-the-box. However, PlanktonFlow also makes it possible to optimize hyperparameters on your own dataset.

Hyperparameter Optimization. Hyperparameter optimization in PlanktonFlow is implemented through Weights & Biases (W&B) sweeps, which provide an efficient and reproducible framework for exploring large hyperparameter spaces. Instead of requiring the user to manually test parameters such as learning rate, weight decay, batch size, or optimizer type, the pipeline automatically launches a series of training runs with different configurations sampled from a predefined search space (specified in the YAML sweep configuration file). The underlying algorithm in W&B prioritizes promising regions of the search space, thereby reducing the number of runs needed compared to a naive grid search.

From the PlanktonFlow user's perspective, initiating a sweep requires only pointing to the provided sweep configuration file. The sweep will continue to launch successive training jobs until it is manually stopped. Typically, users are advised to monitor validation metrics in the W&B dashboard and stop the sweep once these metrics show signs of stable convergence across runs. At this point, the best-performing hyperparameter set can be automatically exported into a dedicated YAML configuration file for further training or inference. PlanktonFlow also saves every trained model together with its configuration, ensuring that optimal runs are reproducible and can be reused in subsequent analyses.

Inference. PlanktonFlow allows batch inference on new image directories. Users can:

- Retrieve top-K predictions per image.
- Apply optional preprocessing (e.g., scale bar removal).
- Export results to CSV for further analysis.

Device-aware execution ensures models run on CPU or GPU depending on availability. Inference outputs include detailed prediction probabilities and optional preprocessing steps applied automatically.

Monitoring and Results Analysis. If Weights & Biases is not used, a custom logging module records training and evaluation metrics in real time. Metrics are stored in CSV files, enabling visualization of training curves, confusion matrices, F1-scores, and class-wise performance. An accompanying Jupyter notebook (`results_analysis.ipynb`) facilitates comprehensive evaluation and figure generation.

Quick Start Example. A complete workflow can be executed as follows:

(1) Preprocess data:

```
python3 run_preprocessing.py --config configs/preprocessing/
    PreprocessWithHierarchical.yaml
```

(2) Train a model:

```
python3 run_training.py --config configs/training/
    TrainDefaultEfficientNet.yaml
```

(3) Make predictions:

```
python3 run_inference.py --config configs/inference/DefaultInference.
    yaml
```

Conclusion

A major strength of PlanktonFlow is its high degree of automation and scalability: all core stages, from data preprocessing to training and inference, are fully automated, minimizing the need for human intervention and making the approach highly suitable for large-scale or continuous monitoring workflows. Also, although the models were trained on a specific plankton dataset, the underlying architecture and modular structure of the pipeline promote generalizability. With only minimal changes to the input format or preprocessing configuration, PlanktonFlow could be adapted to other taxa or image acquisition devices.

Our hope is that the community will use and further develop PlanktonFlow. We foresee that future work could leverage semi-supervised learning, and adapt the pipeline to other imaging and biological systems.

Acknowledgements

The authors thank the Villefranche sur Mer Quantitative Imaging Platform (PIQv) for supporting the community of plankton biologists with free and high-quality services. In particular, our research in general, and this study in particular, rely heavily on Zooprocess and EcoTaxa.

Fundings

Rennes Métropole provided funding to support FlowCAM acquisition by INRAE (project AES 18C0031).

Conflict of interest disclosure

The authors declare that they comply with the PCI rule of having no financial conflicts of interest in relation to the content of the article.

Data, script, code, and supplementary information availability

Datasets are available online at https://doi.org/10.5281/zenodo.16840846 (Gorzerino et al., 2025). Scripts and codes for PlanktonFlow are available at https://github.com/ziraax/PlanktonFlow (Walter, 2025). Supplementary information for figure reproduction and implementation details is available at https://doi.org/10.5281/zenodo.16902839 (Walter et al., 2025).

References

Besson M, Alison J, Bjerge K, Gorochowski TE, Høye TT, Jucker T, Mann HMR, Clements CF (2022). Towards the fully automated monitoring of ecological communities. *Ecology Letters* **25**, 2753–2775. https://doi.org/https://doi.org/10.1111/ele.14123.

Biases W
bibinitperiod (2023). Weights & Biases: experiment tracking, dataset versioning, and model management for machine learning. https://www.wandb.com/.

Dogo EM, Afolabi OJ, Twala B (2022). On the relative impact of optimizers on convolutional neural networks with varying depth and width for image classification. *Applied Sciences* **12**. https://doi.org/10.3390/app122311976.

Eerola T, Batrakhanov D, Barazandeh NV, et al. (2024). Survey of automatic plankton image recognition: challenges, existing solutions and future perspectives. *Artificial Intelligence Review* **57**, 114. https://doi.org/10.1007/s10462-024-10745-y.

Gorzerino C, Walter H, Edeline E, Martignac F (2025). *PlanktonFlow76 - A curated FlowCam dataset for plankton classification*. Zenodo. https://doi.org/10.5281/zenodo.16840846.

He K, Zhang X, Ren S, Sun J (2016). Deep Residual Learning for Image Recognition. In: *2016 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90.

Huang G, Liu Z, van der Maaten L, Weinberger KQ (2018). Densely connected convolutional networks. arXiv: 1608.06993 [cs.CV]. URL: https://arxiv.org/abs/1608.06993.

Irisson JO, Salinas L, Colin S, Complex T, Picheral M (2022). EcoTaxa: a tool to support the taxonomic classification of large datasets through supervised machine learning. In: *SFEcologie 2022*. Metz, France. URL: https://hal.science/hal-04026447.

Jalabert L, Elineau A, Brandão M, Picheral M (2025). *Zooscan-Zooprocess user manual and procedures at the Quantitative Imaging Platform of Villefranche- sur-Mer (PIQv)*. https://doi.org/10.5281/zenodo.15114134.

Khanam R, Hussain M (2024). YOLOv11: an overview of the key architectural enhancements. arXiv: 2410.17725 [cs.CV]. URL: https://arxiv.org/abs/2410.17725.

Lin TY, Goyal P, Girshick R, He K, Dollár P (2018). Focal loss for dense object detection. arXiv: 1708.02002 [cs.CV]. URL: https://arxiv.org/abs/1708.02002.

Maracani A, Pastore VP, Natale L, et al. (2023). In-domain versus out-of-domain transfer learning in plankton image classification. *Scientific Reports* **13**, 10443. https://doi.org/10.1038/s41598-023-37627-7.

Olson RJ, Sosik HM (2007). A submersible imaging-in-flow instrument to analyze nano-and microplankton: Imaging FlowCytobot. *Limnology and Oceanography: Methods* **5**, 195–203. https://doi.org/https://doi.org/10.4319/lom.2007.5.195.

Orenstein EC, Beijbom O, Peacock EE, Sosik HM (2015). WHOI-Plankton- A large scale fine grained visual recognition benchmark dataset for plankton classification. arXiv: 1510.00745 [cs.CV]. URL: https://arxiv.org/abs/1510.00745.

Owen BM, Hallett CS, Cosgrove JJ, Tweedley JR, Moheimani NR (2022). Reporting of methods for automated devices: a systematic review and recommendation for studies using FlowCam for phytoplankton. *Limnology and Oceanography: Methods* **20**, 400–427. https://doi.org/https://doi.org/10.1002/lom3.10496.

Picheral M, Colin S, Irisson JO (2025). EcoTaxa, a tool for the taxonomic classification of images. http://ecotaxa.obs-vlfr.fr. Accessed: 2025-05-29.

Picheral M, Guidi L, Stemmann L, Karl DM, Iddaoud G, Gorsky G (2010). The Underwater Vision Profiler 5: an advanced instrument for high spatial resolution studies of particle size spectra and zooplankton. *Limnology and Oceanography: Methods* **8**, 462–473. https://doi.org/https://doi.org/10.4319/lom.2010.8.462.

Sieracki C, Sieracki M, Yentsch C (1998). An imaging-in-flow system for automated analysis of marine microplankton. *Marine Ecology-progress Series* **168**, 285–296. https://doi.org/10.3354/meps168285.

Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2015). Rethinking the inception architecture for computer vision. arXiv: 1512.00567 [cs.CV]. URL: https://arxiv.org/abs/1512.00567.

Tan M, Le QV (2020). EfficientNet: rethinking model scaling for convolutional neural networks. arXiv: 1905.11946 [cs.LG]. URL: https://arxiv.org/abs/1905.11946.

Walter H (2025). *Deep learning classification pipeline for automatic plankton classification*. Version 1.0.0. URL: https://github.com/ziraax/PlanktonFlow.git.

Walter H, Gorzerino C, Collinet M, Porcon B, Martignac F, Edeline E (2025). PlanktonFlow : supplementary information & Best model weight. https://doi.org/10.5281/zenodo.16902839.

Weinstein BG (2018). A computer vision for animal ecology. *Journal of Animal Ecology* **87**, 533–545. https://doi.org/https://doi.org/10.1111/1365-2656.12780.