Supplementary Information

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

Authors: Walter H., Gorzerino C., Collinet M., Porcon B., Martignac F., Edeline E.
Corresponding author: Eric.Edeline@inrae.fr

S1 Supplementary Methods

S1.1 Loss function algorithms

Algorithm 1 Computation of Focal Loss (per sample)

Require: Logits $\mathbf{o} \in \mathbb{R}^C$, ground-truth label $y \in \{1, \ldots, C\}$, focusing parameter γ, optional weight vector $\boldsymbol{\alpha}$
Ensure: Focal loss $\mathcal{L}_{\text{focal}}$

function $\textsc{FocalLoss}(\mathbf{o}, y, \gamma, \boldsymbol{\alpha})$
  Compute softmax probabilities: $\hat{\mathbf{y}} \leftarrow \text{softmax}(\mathbf{o})$
  Extract true class probability: $p_t \leftarrow \hat{y}_y$
  Clamp p_t to avoid $\log(0)$ (e.g., $p_t \leftarrow \text{clamp}(p_t, \varepsilon, 1-\varepsilon)$)
  Compute modulating factor: $m \leftarrow (1 - p_t)^\gamma$
  if $\boldsymbol{\alpha}$ is defined **then** $a_t \leftarrow \alpha_y$ **else** $a_t \leftarrow 1$
  Compute focal loss: $\mathcal{L} \leftarrow -a_t \cdot m \cdot \log(p_t)$
  return $\mathcal{L}$
end function

Algorithm 2 Computation of Label Smoothing Loss (redistributing smoothing over incorrect classes)

Require: Logits $\mathbf{o}$, ground-truth labels $\mathbf{y}$, smoothing factor ϵ, number of classes C
Ensure: Label smoothing loss $\mathcal{L}_{\text{LS}}$

function $\textsc{LabelSmoothingLoss}(\mathbf{o}, \mathbf{y}, \epsilon)$
  Construct one-hot labels: $\mathbf{y}_{\text{one-hot}}$
  Apply label smoothing:

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

  Compute log-probabilities: $\log \hat{\mathbf{y}} \leftarrow \log \text{softmax}(\mathbf{o})$
  Compute loss: $\mathcal{L} \leftarrow -\sum_{c=1}^{C} \tilde{y}_c \cdot \log \hat{y}_c$
  return $\mathcal{L}$
end function

S1.2 Learning Rate Scheduling

In addition to optimizer selection, the learning rate was dynamically adjusted throughout training using a cosine annealing schedule. Specifically, we used PyTorch's `CosineAnnealingLR`, which gradually decreases the learning rate following a cosine function over a fixed number of epochs. This approach encourages larger updates at the beginning of training and finer adjustments toward the end, helping the model settle into better minima.

At each epoch t, the learning rate η_t is computed as:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min}) \left(1 + \cos\left(\frac{t\pi}{T_{\max}}\right)\right) \tag{S1}$$

where η_0 is the initial learning rate, $\eta_{\min}$ is the minimum learning rate (often set to 0), t is the current epoch, and $T_{\max}$ is the total number of epochs before a full cosine cycle completes.

S1.3 Early Stopping Strategy

To prevent unnecessary training time, an **early stopping** mechanism was implemented (see Algorithm 3). This method monitors the validation loss and, optionally, the validation accuracy, halting training when no improvement is observed for a specified number of consecutive epochs (referred to as the *patience* parameter).

Our implementation allows for two simultaneous criteria:

- Validation loss must improve by at least a threshold δ (default: 0).
- If enabled, validation accuracy must also improve by at least δ.

Formally, let $\mathcal{L}_t$ and $\mathcal{A}_t$ denote the validation loss and accuracy at epoch t. Training stops if:

$$\forall\, k \in \{1, \dots, \text{patience}\}, \quad \mathcal{L}_{t-k} \geq \mathcal{L}_{t-k-1} - \delta \quad \text{and} \quad \mathcal{A}_{t-k} \leq \mathcal{A}_{t-k-1} + \delta$$

This dual-metric early stopping criterion improves robustness over single-metric strategies, particularly in settings where accuracy and loss are not perfectly aligned.

Algorithm 3 Early Stopping Procedure

Require: Validation loss $\mathcal{L}_t$, validation accuracy $\mathcal{A}_t$, patience p, improvement threshold δ, model M

Initialize: best_loss $\leftarrow \infty$, best_acc $\leftarrow 0$, counter $\leftarrow 0$
for each epoch t **do**
    Evaluate current $\mathcal{L}_t$, $\mathcal{A}_t$
    improved $\leftarrow$ **False**
    if $\mathcal{L}_t <$ best_loss $- \delta$ **then**
        best_loss $\leftarrow \mathcal{L}_t$
        improved $\leftarrow$ **True**
    end if
    if monitor accuracy **and** $\mathcal{A}_t >$ best_acc $+ \delta$ **then**
        best_acc $\leftarrow \mathcal{A}_t$
        improved $\leftarrow$ **True**
    end if
    if improved **then**
        counter $\leftarrow 0$
        Save model checkpoint M
    else
        counter $\leftarrow$ counter $+ 1$
        if counter $\geq p$ **then**
            Stop training
        end if
    end if
end for

S2 Additional Figures

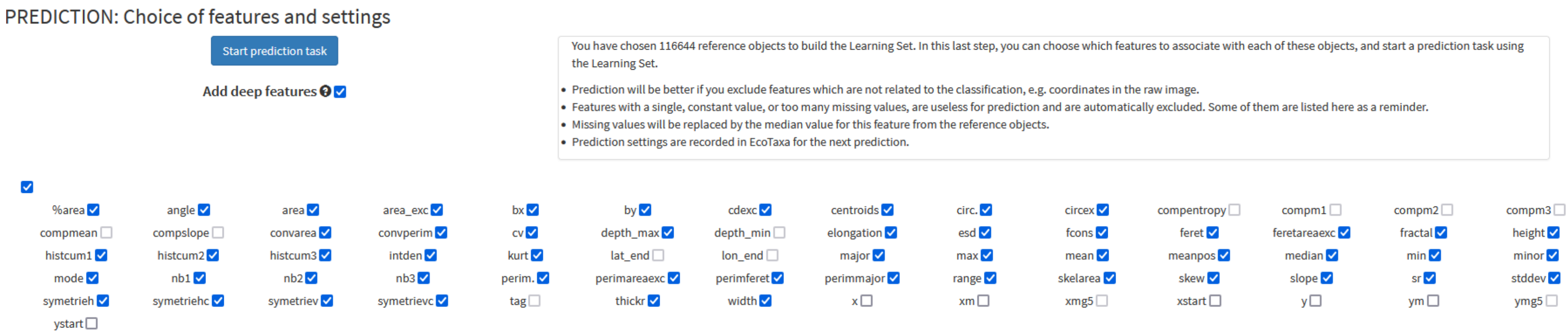

Figure S1: EcoTaxa selected parameters

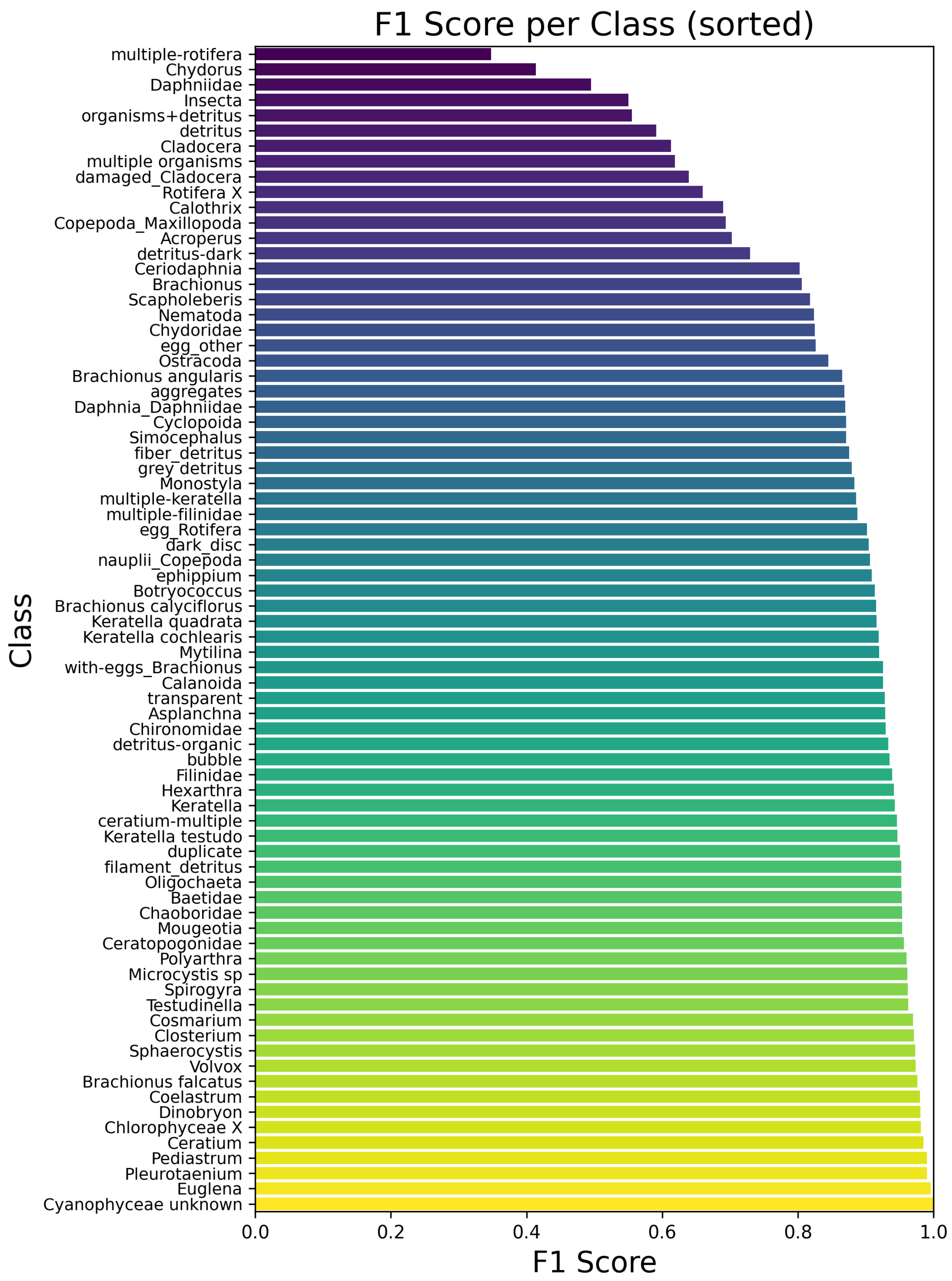

Figure S2: F1 Score per class for the best trained model (EfficientNet-B5)