Pipeline: AI assisted labeling of IFCB data with LabelChecker

Kananat Siangsano

June 1, 2025

1 Introduction

This document describes a pipeline for processing Imaging FlowCytobot (IFCB) data and training YOLO classification models for automated plankton identification. The pipeline consists of four main components that work together to convert raw IFCB data into a format suitable for machine learning, facilitate ai-assisted labeling, and iteratively improve classification accuracy.

The complete workflow enables users to:

- Convert raw IFCB data (.roi, .adc, .hdr files) into a format compatible with LabelChecker (`https://github.com/TimWalles/LabelChecker`)
- Create properly structured datasets for Ultralytics YOLO model training
- Train classification models using Ultralytics YOLO (`https://www.ultralytics.com/`)
- Apply trained models to predict labels and iteratively improve accuracy

2 Pipeline Components

2.1 IFCB Preprocessing (lcifcb_preprocessing.py)

2.1.1 Overview

The `lcifcb_preprocessing.py` script converts raw IFCB data files into a format that can be read and processed by LabelChecker software. IFCB instruments generate three types of files for each sample:

- **.roi files**: Contain raw image data as binary arrays
- **.adc files**: Contain metadata and region of interest coordinates
- **.hdr files**: Contain header information (optional)

2.1.2 Key Features

- Extracts individual plankton images from binary ROI data
- Generates CSV metadata compatible with LabelChecker
- Creates organized directory structure for processed data
- Handles multiple sample files in batch processing mode

2.1.3 Usage Example

Listing 1: Basic usage

```
# Process IFCB files to LabelChecker format
python lcifcb_preprocessing.py /path/to/ifcb/data -o /path/to/output

# Verbose output
python lcifcb_preprocessing.py /path/to/ifcb/data -o /path/to/output -v
```

2.1.4 Input and Output Format

Expected Input Structure:

```
input_folder/
          sample_001.roi    # Binary image data
          sample_001.adc    # Metadata and coordinates
          sample_001.hdr    # Header information (optional)
          sample_002.roi
          sample_002.adc
          sample_002.hdr
```

Generated Output Structure:

```
output_folder/
          sample_001/
                sample_001/           # Individual images
                      sample_001_00000.png
                      sample_001_00001.png
                      ...
                LabelChecker_sample_001.csv
          sample_002/
                sample_002/
                      sample_002_00000.png
                      sample_002_00001.png
                      ...
                LabelChecker_sample_002.csv
```

2.2 Training Dataset Creation (create_training_dataset.py)

2.2.1 Overview

The `create_training_dataset.py` script converts labeled data from LabelChecker CSV files into the directory structure required by YOLO classification models. It implements stratified splitting to ensure balanced representation of all classes across training, validation, and test sets.

2.2.2 Key Features

- Stratified train/validation/test splitting maintaining class distribution
- Configurable split ratios (default: 70% train, 15% validation, 15% test)
- Automatic handling of classes with minimal samples
- Batch processing of multiple LabelChecker CSV files
- Statistics and error reporting

2.2.3 Usage Example

Listing 2: Dataset creation usage

```
# Create dataset with default ratios (70/15/15)
python create_training_dataset.py --input_folder /path/to/labelchecker/data \
                                  --output_folder /path/to/yolo/dataset

# Custom split ratios
python create_training_dataset.py --input_folder /path/to/labelchecker/data \
                                  --output_folder /path/to/yolo/dataset \
                                  --train_ratio 0.8 --val_ratio 0.1 --
                                      test_ratio 0.1

# Set random seed for reproducibility
python create_training_dataset.py --input_folder /path/to/labelchecker/data \
                                  --output_folder /path/to/yolo/dataset \
                                  --random_state 123
```

The script automatically searches for all `LabelChecker_*.csv` files in the input directory and processes them in batch mode.

2.3 Model Training (training.ipynb)

2.3.1 Overview

The `training.ipynb` Jupyter notebook provides a minimum script for training YOLO classification models. It utilizes the Ultralytics YOLO framework with the YOLOv11 medium classification model as the base architecture.

The trained model should be saved to runs/classify/train/weights/best.pt

2.3.2 Training Parameters

- **Base Model**: YOLOv11 medium classification (`yolo11m-cls.pt`)
- **Image Size**: 224×224 pixels (standard for classification tasks)
- **Batch Size**: 32 (adjustable based on GPU memory)
- **Epochs**: 500 (with early stopping based on validation performance)
- **Device**: GPU acceleration recommended for faster training

2.4 Model Prediction (model_prediction.py)

2.4.1 Overview

The `model_prediction.py` script applies trained YOLO models to predict labels for unlabeled images in LabelChecker CSV files. It updates the `LabelPredicted` column in LabelChecker*.csv with model predictions and `ProbabilityScore` with confidence scores.

2.4.2 Key Features

- Batch processing of multiple CSV files
- Configurable confidence threshold filtering
- Selective processing (only images without existing verified labels)

- Progress reporting and error handling
- Preservation of existing manual labels

2.4.3 Usage Example

Listing 3: Model prediction usage

```
# Basic prediction with default threshold (0.8)
python model_prediction.py --input_folder /path/to/labelchecker/data \
                           --model_path /path/to/trained/model.pt

# Custom confidence threshold = 0.9
python model_prediction.py --input_folder /path/to/labelchecker/data \
                           --model_path /path/to/trained/model.pt \
                           --threshold 0.9
```

3 Complete Workflow Example

This section demonstrates the complete pipeline from raw IFCB data to a trained classification model.

3.1 Step 1: Data Preprocessing

Convert raw IFCB data to LabelChecker format:

```
# Process raw IFCB files
python lcifcb_preprocessing.py /data/raw_ifcb_samples -o /data/
    labelchecker_format
```

3.2 Step 2: Manual Labeling

Use LabelChecker software to manually label a subset of images:

1. Open LabelChecker and load the generated CSV files
2. Create classification categories (e.g., diatoms, dinoflagellates, copepods)
3. Label representative samples from each category (Ideally at least 100 images for each class)

3.3 Step 3: Dataset Creation and Training

Create YOLO-compatible dataset and train the model:

```
# Create training dataset
python create_training_dataset.py --input_folder /data/labelchecker_format \
                                  --output_folder /data/yolo_dataset \
                                  --train_ratio 0.7 --val_ratio 0.15 --
                                      test_ratio 0.15
```

And then train the model using Jupyter notebook trining.ipynb

3.4 Step 4: Model Application

Apply the trained model to predict labels for remaining unlabeled images:

```
# Apply model predictions
python model_prediction.py --input_folder /data/labelchecker_format \
                           --model_path /data/models/best.pt \
                           --threshold 0.8

# Review predictions in LabelChecker
# Correct any misclassifications and add to training data
```

3.5 Step 5: Iterative Improvement

Repeat the cycle to improve model accuracy:

1. Review model predictions in LabelChecker
2. Correct misclassifications and add difficult cases to training set
3. Re-run dataset creation with updated labels
4. Retrain model with expanded dataset
5. Apply improved model to remaining data
6. Continue until satisfactory accuracy is achieved

4 Troubleshooting

4.1 Common Issues

File Path Problems:

- Ensure all file paths use forward slashes or raw strings
  (Add prefix r to path: [path = r"path/to/data"] instead of [path = "path/to/data"])
- Verify that input directories contain expected file types/structure
- Check file permissions for read/write access

Memory Issues:

- Reduce batch size if encountering CUDA out-of-memory errors

Training Problems:

- Verify dataset structure matches YOLO requirements
- Check for class imbalance issues
- Ensure sufficient training data for each class

5 Conclusion

This pipeline provides a minimalistic solution for automated plankton classification using IFCB data with LabelChecker and YOLO models. The iterative improvement process enables continuous refinement of classification accuracy as more data becomes available.

The combination of automated preprocessing, manual validation, and machine learning prediction creates an efficient workflow that can significantly reduce the time required for large-scale plankton identification tasks while maintaining scientific rigor and accuracy.