Algae Microscopy Classifier

CSE381: Introduction to Machine Learning
Project 44

GitHub repo

MEMBERS

Adham Hisham Kandil

22P0217

Ahmed Mohamed Lotfy

22P0251

Represented to

Dr. Alaa Mahmoud Hamdy
Professor, Computer and Systems Engineering

Eng. Engy Ahmed
Eng. George Emad
Teaching Assistant, Computer and Systems Engineering

Contents

1. Executive Summary

1.1 Project Overview

This project presents a comprehensive machine learning system for automated classification of algae species from microscopic images. The system implements a two-stage pipeline combining object detection (YOLO) with classification using both traditional machine learning and deep learning approaches. The goal is to assist biological and environmental analysis by accurately identifying six different algae species: Platymonas, Chlorella, Dunaliella salina, Effrenium, Porphyridium, and Haematococcus.

1.2 Key Achievements

The project successfully implemented a dual-approach machine learning system that combines both traditional machine learning and deep learning methodologies. The traditional ML approach incorporates multiple classifiers including XGBoost, Random Forest, Decision Tree, and Artificial Neural Networks (ANN), while the deep learning approach utilizes Convolutional Neural Networks (CNN) with transfer learning capabilities. This dual implementation allows for comprehensive comparison and evaluation of different machine learning paradigms on the same dataset.

The evaluation framework implemented in this project is particularly robust, incorporating multiple evaluation strategies to provide a comprehensive assessment of model performance. The system includes both ground truth-based evaluation, which assumes perfect object detection, and YOLO-based evaluation, which simulates real-world scenarios by using actual object detection outputs. This dual evaluation approach provides both an upper bound estimate of performance and a realistic assessment of how the system would perform in practical applications.

The best performance achieved in this project is 75.66% accuracy on the test set using the XGBoost classifier with the traditional machine learning approach. This result demonstrates that carefully engineered handcrafted features can outperform deep learning methods when training data is limited. The XGBoost model achieved this performance through systematic hyperparameter tuning and feature selection, resulting in a well-optimized classifier that effectively handles the class imbalance present in the dataset.

2. Introduction

2.1 Problem Statement

Manual identification and classification of algae species from microscopic images is a time-consuming and labor-intensive process that requires specialized expertise. With the increasing need for environmental monitoring and algae-based research, there is a critical demand for automated, accurate, and scalable classification systems. This project addresses this challenge by developing a machine learning-based solution that can automatically detect and classify algae species from microscopy images.

2.2 Project Scope and Goals

The primary goal of this project is to develop a fully automated system for algae species classification from microscopy images that can accurately identify six different algae species with minimal human intervention. This system aims to address the time-consuming nature of manual identification while maintaining high accuracy standards required for scientific and commercial applications. The development process emphasizes creating a robust, reproducible, and extensible solution that can serve as a foundation for future enhancements and applications.

Model performance evaluation is conducted using multiple metrics and strategies to provide a comprehensive assessment of system capabilities. The evaluation framework includes accuracy, precision, recall, and F1-score metrics, calculated both as overall measures and per-class measures. Additionally, the system implements two evaluation strategies: ground truth-based evaluation that assumes perfect object detection, and YOLO-based evaluation that simulates real-world performance by incorporating detection errors. This dual evaluation approach provides both optimistic and realistic performance estimates.

The project scope encompasses the classification of six algae species: Platymonas, Chlorella, Dunaliella salina, Effrenium, Porphyridium, and Haematococcus. The system implements a two-stage pipeline that first detects algae instances in images using YOLO object detection, then classifies each detected instance using either traditional machine learning or deep learning approaches. This two-stage design allows the system to handle images containing multiple algae instances and provides flexibility in choosing the classification methodology. The scope includes comprehensive evaluation and visualization capabilities that enable detailed analysis of model performance, feature importance, and classification patterns.

2.3 Dataset Source

The project uses the **High-throughput Algae Cell Detection** dataset from Kaggle (marquis03/high-throughput-algae-cell-detection). The dataset contains microscopy images with YOLO-format annotations, providing bounding boxes and class labels for algae instances in each image.

3. Dataset Description

3.1 Dataset Structure

The dataset is organized in YOLO format with the following structure:70% training | 30% testing

```
├── train/
│  ├── images/ # 700 training images (.jpg)
│  └── labels/ # 700 corresponding label files (.txt)
├── test/
│  ├── images/ # 300 test images (.jpg)
│  └── labels/ # 300 corresponding label files (.txt)
└── data.yaml # Dataset configuration
```

3.2 Data Format

Image Format:

File type: JPEG (.jpg)
Color space: RGB
Variable dimensions (resized to 224Ã—224 for training)

Label Format (YOLO): Each label file contains one line per object with the format:

class_id x_center y_center width height
All coordinates are normalized (0-1 range).

Example label file:
0 0.523 0.456 0.123 0.098
1 0.789 0.234 0.045 0.067

3.3 Class Distribution

The dataset contains 6 algae species with the following class IDs and names:

Class ID	*Species Name*	*Training Instances*
0	*Platymonas*	*184*
1	*Chlorella*	*722*
2	*Dunaliella salina*	*234*
3	*Effrenium*	*227*
4	*Porphyridium*	*272*
5	*Haematococcus*	*119*

Total Training Instances: 1,758 objects across 700 images

3.4 Class Imbalance Analysis

The dataset exhibits significant class imbalance that presents substantial challenges for model training and evaluation. Chlorella (class 1) is the most represented species with 722 instances, accounting for 41.1% of all training instances. This dominance means that the model will see Chlorella examples much more frequently during training, potentially leading to a bias toward predicting this class. In contrast, Haematococcus (class 5) is the least represented species with only 119 instances, comprising just 6.8% of the training data. This creates an imbalance ratio of approximately 6:1 between the most and least represented classes.

The intermediate classes show more balanced representation: Dunaliella salina has 234 instances (13.3%), Effrenium has 227 instances (12.9%), Porphyridium has 272 instances (15.5%), and Platymonas has 184 instances (10.5%). However, the significant disparity between Chlorella and Haematococcus, combined with the overall uneven distribution, creates a challenging learning scenario where the model must learn to distinguish between classes with vastly different amounts of training data.

This class imbalance presents several challenges for model training. Models trained on imbalanced data often develop a bias toward the majority class, leading to poor performance on minority classes. The model may achieve high overall accuracy by simply predicting the majority class most of the time, but this strategy fails to provide useful classifications for the underrepresented species. Additionally, the limited number of examples for minority classes makes it difficult for the model to learn the distinguishing characteristics of these species.

3.5 Data Characteristics

Image Statistics:

1. Training images: 700
2. Test images: 300
3. Average objects per image: ~2.5 (training set)

Bounding Box Statistics:

1. Normalized dimensions range from 0.01 to 0.5
2. Average aspect ratios vary by species
3. Size distributions show species-specific patterns

4. Methodology

4.1 Two-Stage Pipeline Architecture

The system employs a two-stage approach:

1. **Stage 1: Object Detection (YOLO)**
2. Detects algae instances in microscopy images
3. Outputs bounding boxes for each detected object
4. Provides confidence scores for detections
5. **Stage 2: Classification**
6. **Traditional ML Approach**: Extracts handcrafted features and uses ML classifiers
7. **Deep Learning Approach**: Uses CNN for end-to-end classification

4.2 Traditional Machine Learning Approach

4.2.1 Feature Extraction Methodology

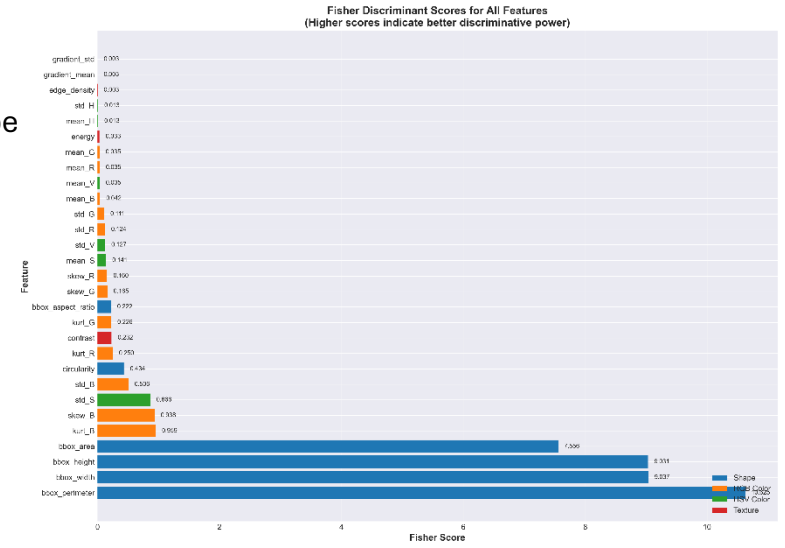
The traditional ML approach relies on handcrafted features extracted from detected algae regions, capturing morphological, color, and texture characteristics. Size features include bounding box width, height, area, perimeter, aspect ratio, and circularity (4Ï€ Ã— area / perimeterÂ²), providing measures of size and shape. Color features are extracted in both RGB and HSV color spaces, computing mean, standard deviation, skewness, and kurtosis for RGB channels, and mean and standard deviation for HSV channels. Texture features include contrast (standard deviation of grayscale values), energy (normalized sum of squared pixel values), edge density (proportion of edge pixels), and gradient magnitude statistics from Sobel operators. In total, 24 features are extracted per algae object, creating a comprehensive numerical representation for machine learning classifiers.

4.2.2 Fisher Discriminant Analysis for Feature Selection

Fisher Discriminant Analysis (FDA) is used to rank features by their discriminative power. The Fisher score measures the ratio of between-class variance to within-class variance:

Top 8 Selected Features (by Fisher score):

1. `bbox_perimeter` (Score: 10.62) - Shape
2. `bbox_width` (Score: 9.04) - Shape
3. `bbox_height` (Score: 9.03) - Shape
4. `bbox_area` (Score: 7.56) - Shape
5. `kurt_B` (Score: 0.96) - RGB Color
6. `skew_B` (Score: 0.94) - RGB Color
7. `std_S` (Score: 0.87) - HSV Color
8. `std_B` (Score: 0.51) - RGB Color



Shape features dominate the top rankings, indicating that morphological characteristics are most discriminative for algae classification.

4.2.3 Classifiers Implemented

Four different machine learning classifiers are implemented and evaluated to determine the most effective approach for algae species classification. Each classifier represents a different machine learning paradigm with distinct characteristics, advantages, and limitations.

Decision Tree, Random Forest, XGBoost, and MLP (Multi-Layer Perceptron). Decision Tree is interpretable but prone to overfitting, using hyperparameters max_depth, min_samples_split, and min_samples_leaf. Random Forest creates an ensemble of trees trained on different data subsets, reducing overfitting and handling class imbalance better, with key hyperparameters n_estimators, max_depth, and min_samples_split. XGBoost uses sequential gradient boosting where each tree corrects previous errors, offering built-in regularization and feature importance rankings, with hyperparameters n_estimators, learning_rate, max_depth, and subsample. MLP uses feedforward neural networks with hidden layers to learn non-linear boundaries, requiring careful hyperparameter tuning (hidden_layer_sizes, learning_rate, alpha) and feature scaling.

4.2.5 Model Selection Criteria

Models are evaluated using multiple metrics:

1. **Accuracy**: Overall correctness
2. **Precision**: Weighted average precision across classes
3. **Recall**: Weighted average recall across classes
4. **F1-Score**: Harmonic mean of precision and recall

The best model is selected based on test set accuracy.

4.3 Deep Learning Approach

4.3.1 CNN Architecture

The deep learning approach uses transfer learning with pretrained CNN architectures:

Base Architectures:

1. **EfficientNet-B0**: Efficient, mobile-friendly architecture
2. **ResNet18**: Residual network with 18 layers
3. **Vision Transformer (ViT)**: Transformer-based architecture (optional)

Model Configuration:

1. Input size: 224Ã—224Ã—3 (RGB)
2. Pretrained weights: ImageNet
3. Custom classifier head:
4. Dropout (0.3-0.5)
5. Fully connected layer (512 units)
6. ReLU activation
7. Dropout
8. Output layer (6 classes)

4.3.2 Transfer Learning Strategy

1. **Freeze Backbone**: Initially freeze pretrained layers
2. **Fine-tune Classifier**: Train only the custom classifier head
3. **Unfreeze and Fine-tune**: Gradually unfreeze and fine-tune entire network
4. **Learning Rate Scheduling**: Reduce learning rate during fine-tuning

4.3.3 Data Augmentation Pipeline

Critical for microscopy images to improve generalization:

Training Augmentations:

1. Horizontal flip (p=0.5)
2. Vertical flip (p=0.5)
3. Random rotation (90Â° increments, p=0.5)
4. Brightness adjustment (Â±20%)
5. Contrast adjustment (Â±20%)
6. Gaussian noise (var_limit=0.01, p=0.3)

Validation/Test:

1. Resize to 224Ã—224
2. Normalization only (ImageNet statistics)

4.3.4 Training Procedure

Hyperparameters:

1. Batch size: 16-32
2. Learning rate: 0.001-0.0001
3. Optimizer: Adam
4. Loss function: CrossEntropyLoss
5. Epochs: 50 (with early stopping)
6. Early stopping patience: 10 epochs
7. Weight decay: 0.0001

Training Strategy:

1. Load pretrained weights
2. Replace classifier head
3. Train with frozen backbone (10 epochs)
4. Unfreeze and fine-tune (remaining epochs)
5. Save best model based on validation accuracy

4.3.5 Loss Functions and Optimization

Loss Function:

1. CrossEntropyLoss with optional class weights for imbalance

Optimizer:

1. Adam optimizer with:
2. Learning rate: 0.0001
3. Weight decay: 0.0001 (L2 regularization)
4. Beta1: 0.9, Beta2: 0.999

Learning Rate Scheduling:

1. ReduceLROnPlateau: Reduce LR by factor of 0.1 when validation loss plateau

5. Implementation Details

5.1 Data Pipeline

5.1.1 Image Preprocessing

Standard Preprocessing:

1. Load image using OpenCV (BGR format)
2. Convert to RGB
3. Resize to target size (224Ã—224 for CNNs, variable for feature extraction)
4. Normalize pixel values to [0, 1] range
5. Apply ImageNet normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

Code Example:

```
import cv2
import albumentations as A
from albumentations.pytorch import ToTensorV2

transform = A.Compose([
    A.Resize(224, 224),
    A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ToTensorV2()
])
```

5.1.2 YOLO Format Label Parsing

Labels are parsed from text files with format: `class_id x_center y_center width height`

Implementation:

```python
def _parse_yolo_label(self, label_path: Path) -> Tuple[List, List]:
    boxes = []
    class_ids = []

    with open(label_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            class_id = int(parts[0])
            x_center = float(parts[1])
            y_center = float(parts[2])
            width = float(parts[3])
            height = float(parts[4])

            boxes.append([x_center, y_center, width, height])
            class_ids.append(class_id)

    return boxes, class_ids
```

5.1.3 Data Augmentation Strategies

Albumentations Pipeline:

1. Handles both image and bounding box transformations
2. Maintains YOLO format during augmentation
3. Removes boxes with <30% visibility after augmentation

Augmentation Configuration:

```yaml
augmentation:
  train:
    horizontal_flip: 0.5
    vertical_flip: 0.5
    rotation: 90
    brightness: 0.2
    contrast: 0.2
    gaussian_noise: 0.01
```

5.1.4 Batch Processing

Custom Collate Function: Handles variable number of objects per image:

```
def collate_fn(batch):
    images, targets, metadata = zip(*batch)
    images = torch.stack(images, 0)

    # Update batch indices in targets
    for i, target in enumerate(targets):
        target[:, 0] = i
    targets = torch.cat(targets, 0)

    return images, targets, list(metadata)
```

5.2 Feature Engineering

5.2.1 Feature Extraction Functions

Color Features:

```
def extract_color_features(image: np.ndarray) -> Dict[str, float]:
    features = {}

    # RGB statistics
    for i, channel in enumerate(['R', 'G', 'B']):
        features[f'mean_{channel}'] = np.mean(image[:, :, i])
        features[f'std_{channel}'] = np.std(image[:, :, i])
        features[f'skew_{channel}'] = pd.Series(image[:, :, i].flatten()).skew()
        features[f'kurt_{channel}'] = pd.Series(image[:, :,
i].flatten()).kurtosis()

    # HSV statistics
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    for i, channel in enumerate(['H', 'S', 'V']):
        features[f'mean_{channel}'] = np.mean(hsv[:, :, i])
        features[f'std_{channel}'] = np.std(hsv[:, :, i])

    return features
```

Shape Features:

```
def extract_shape_features(image: np.ndarray, bbox: List[float]) -> Dict[str,
float]:
    h, w = image.shape[:2]
    x_center, y_center, box_w, box_h = bbox

    box_w_px = box_w * w
    box_h_px = box_h * h

    features = {
        'bbox_width': float(box_w_px),
        'bbox_height': float(box_h_px),
        'bbox_area': float(box_w_px * box_h_px),
        'bbox_aspect_ratio': float(box_w_px / (box_h_px + 1e-6)),
        'bbox_perimeter': float(2 * (box_w_px + box_h_px)),
        'circularity': float(4 * np.pi * area / (perimeter ** 2 + 1e-6))
    }

    return features
```

5.2.2 Feature Selection Using Fisher Scores

Fisher scores are calculated for all features, and top-ranked features are selected based on discriminative power. The selection process:

1. Extract all features from training set
2. Calculate Fisher score for each feature
3. Rank features by Fisher score
4. Select top N features (typically 8-10)
5. Save selected features to CSV for reproducibility

5.3 Model Training

5.3.1 Traditional ML Training Procedure

1. **Load and preprocess data**
2. Extract features from training set
3. Select features using Fisher scores
4. Scale features
5. **Hyperparameter tuning**
6. GridSearchCV with 5-fold cross-validation
7. Evaluate on validation set
8. **Train final model**
9. Train on full training set with best hyperparameters
10. Evaluate on test set
11. **Save model and scaler**
12. Save trained model (pickle)
13. Save feature scaler

5.3.2 Deep Learning Training Procedure

1. **Setup**
    1. Load pretrained model
    2. Replace classifier head
    3. Setup optimizer and loss function
2. **Training loop**

```
for epoch in range(num_epochs):
    # Training phase
    model.train()
    for images, targets, metadata in train_loader:
        # Forward pass
        # Backward pass
        # Optimizer step

    # Validation phase
    model.eval()
    with torch.no_grad():
        # Evaluate on validation set

    # Early stopping check
    # Save best model
```

3. **Fine-tuning**

    1. Freeze backbone initially
    2. Train classifier head
    3. Unfreeze and fine-tune entire network

5.3.3 Validation Strategy

1. **Traditional ML**: 5-fold cross-validation during hyperparameter tuning
2. **Deep Learning**: Separate validation set (if available) or train/val split
3. **Early Stopping**: Monitor validation loss, stop if no improvement for N epochs

5.3.4 Model Checkpointing

1. Save best model based on validation accuracy
2. Save training state (optimizer, epoch, best accuracy)
3. Load checkpoint for resuming training or inference

5.4 Evaluation Framework

5.4.1 Metrics Used

Classification Metrics:

1. **Accuracy**: Overall correctness
2. **Precision** (weighted): Average precision across classes
3. **Recall** (weighted): Average recall across classes
4. **F1-Score** (weighted): Harmonic mean of precision and recall

Per-Class Metrics:

1. Precision, recall, F1-score for each algae species
2. Support (number of instances per class)

5.4.2 Confusion Matrix Analysis

Confusion matrices visualize:

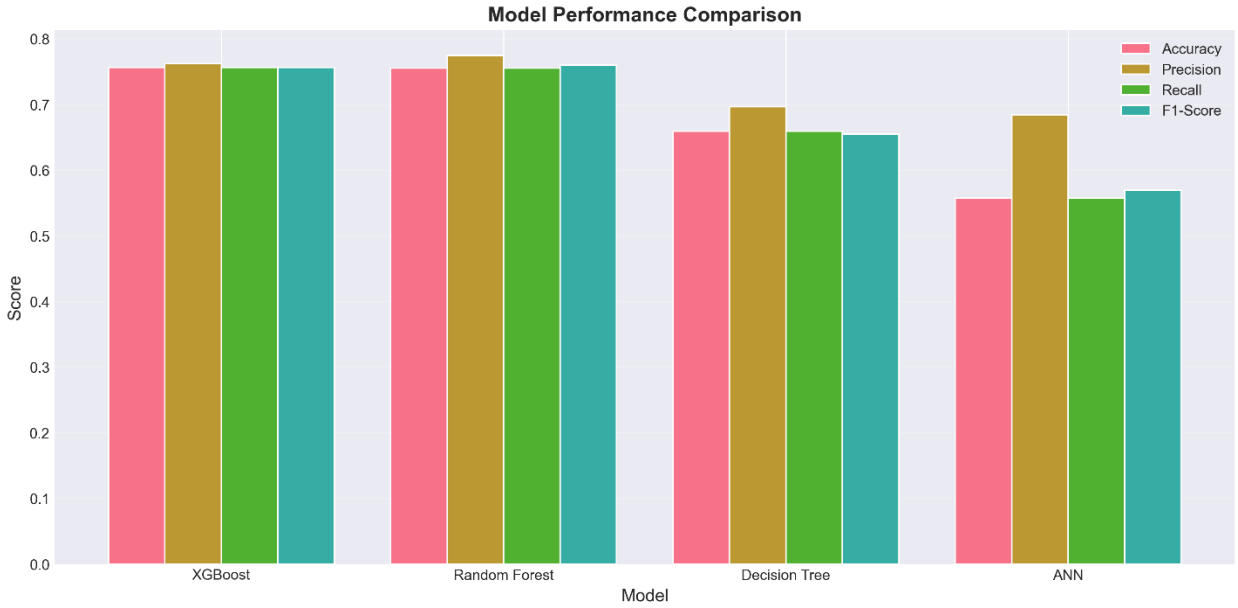
1. True positives, false positives, false negatives per class
2. Class-wise performance
3. Common misclassification patterns

6. Experimental Results

6.1 Traditional ML Results

6.1.1 Model Comparison

Evaluation on the test set (300 images, 1,627 instances) reveals significant performance differences. XGBoost achieved the best performance with 75.66% accuracy, 76.21% precision, 75.66% recall, and 75.62% F1-score, demonstrating consistent performance across metrics. Random Forest achieved very similar results (75.54% accuracy, 77.47% precision, 75.54% recall, 76.03% F1-score), representing a strong alternative. Decision Tree achieved moderate performance (65.95% accuracy), expectedly lower than ensemble methods due to overfitting tendencies. MLP achieved the lowest performance (55.75% accuracy), likely due to limited training data and suboptimal feature representation for neural networks. The clear hierarchy demonstrates the value of ensemble methods, with XGBoost recommended as the best choice.



7.1.2 Best Model Performance (XGBoost)

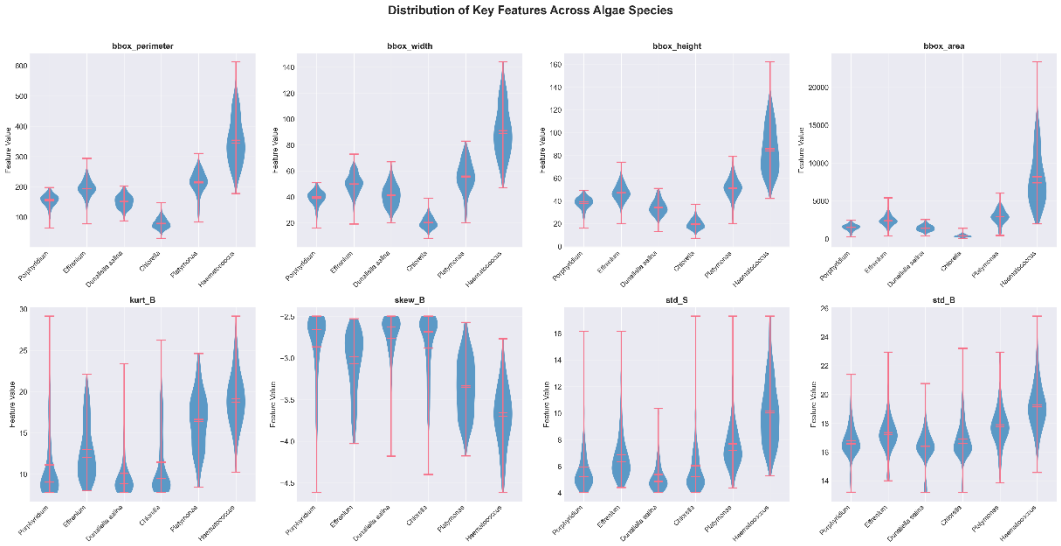
XGBoost achieves 75.66% accuracy, correctly classifying approximately three out of every four algae instances despite class imbalance challenges. The precision of 76.21% indicates high reliability when making predictions, valuable for quality control applications. The recall of 75.66% and F1-score of 75.62% show balanced performance between identifying species and avoiding false positives. Optimal hyperparameters from grid search include n_estimators=200, learning_rate=0.1, max_depth=5, and subsample=0.8, representing a well-balanced configuration that maximizes performance while preventing overfitting.

6.1.3 Feature Importance Analysis

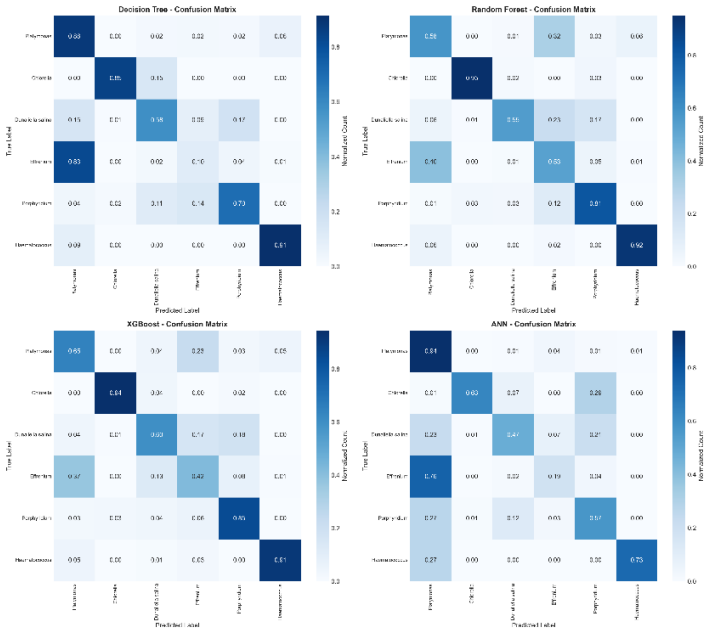
Top features by importance (XGBoost):

1. `bbox_perimeter` (highest importance)
2. `bbox_width`
3. `bbox_height`
4. `bbox_area`
5. Color features (kurt_B, skew_B, std_S, std_B)

Shape features dominate, confirming Fisher score rankings.



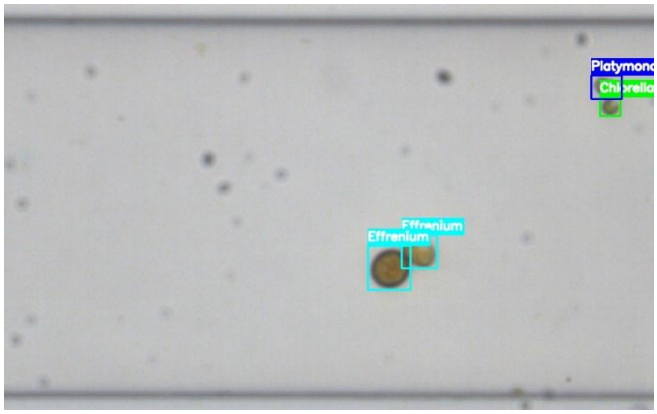
6.1.4 Confusion Matrices



Confusion matrices reveal:

1. **Chlorella** (class 1): Best classification performance (most training data)
2. **Haematococcus** (class 5): Lower performance (least training data, class imbalance)
3. Common confusions between morphologically similar species

6.2 Deep Learning Results



6.2.1 Training Curves

Best Configuration:

Architecture: ResNet18
Batch size: 32
Learning rate: 0.0001
Test Accuracy: 44.00%

Training Observations:

1. Validation accuracy plateaus around epoch 20-30
2. Early stopping prevents overfitting
3. Lower performance compared to traditional ML (likely due to limited training data)

6.2.2 Validation Performance

1. Best validation accuracy: ~44%
2. Training loss decreases steadily
3. Validation loss plateaus early (indicates need for more data or different augmentation)

6.2.3 Test Set Results

CNN Performance:

1. Test Accuracy: 44.00%
2. Lower than traditional ML approach
3. Possible reasons:
4. Limited training data (700 images)
5. Class imbalance
6. Need for more aggressive augmentation
7. Different architecture or hyperparameters

6.2.4 Model Comparison

Hyperparameter Grid Search Results:

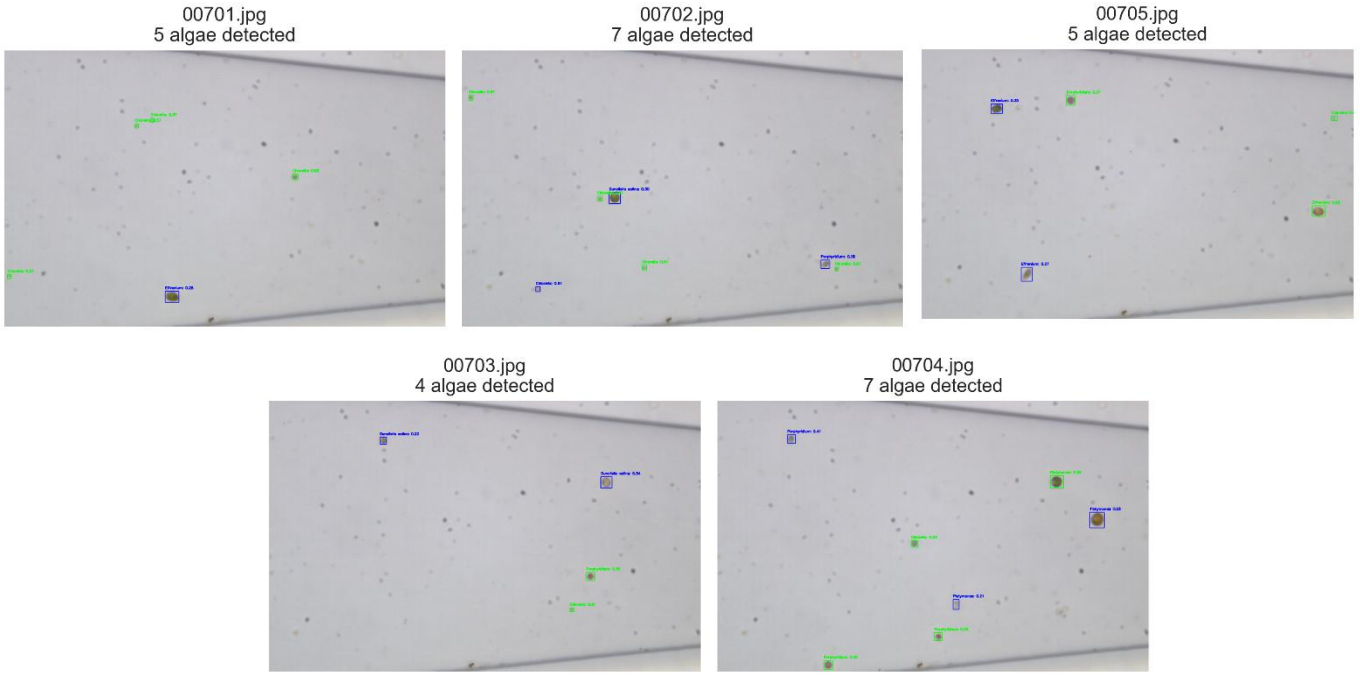
1. Tested 16 combinations (batch_size Ã— learning_rate)
2. Best: batch_size=32, learning_rate=0.0001
3. Accuracy range: 12% - 44%

6.3 Comparative Analysis

6.3.1 Traditional ML vs. Deep Learning Performance

Traditional ML (XGBoost) achieved 75.66% accuracy, substantially outperforming the deep learning CNN approach (44.00% accuracy). The performance gap of over 30 percentage points is primarily due to limited training data (700 images), which is insufficient for deep learning models that typically require tens of thousands of images. Traditional ML advantages include better performance with limited data, interpretable features (particularly shape features that are highly discriminative), faster training (minutes vs. hours), and lower computational requirements. However, it requires manual feature engineering and may miss complex patterns.

Demo – Final results sample



7. Future Work

7.1 Potential Improvements

Several improvements can be made to enhance the overall performance and robustness of the system. From a data perspective, increasing the size of the training dataset to more than 5,000 images per class would significantly improve generalization. Ensuring higher data quality and consistency, expanding the dataset to include additional algae species, and incorporating images captured under diverse environmental conditions would further strengthen model reliability.

In terms of feature engineering, exploring richer feature representations such as texture and morphological features could improve discriminative power. Automated feature selection techniques and modeling feature interactions may also help identify the most informative attributes.

On the model side, experimenting with advanced CNN architectures such as EfficientNet variants and Vision Transformers, using ensemble methods that combine traditional machine learning and deep learning models, and integrating attention mechanisms could lead to more effective feature learning.

7.2 Additional Features to Explore

Future work can benefit from incorporating advanced feature types that capture more detailed image characteristics. Texture-based features such as Gray-Level Co-occurrence Matrix (GLCM) and Local Binary Patterns (LBP) can help represent surface patterns, while morphological features derived from contour analysis can capture shape-related information.

Additionally, frequency-domain features obtained using Fast Fourier Transform (FFT) or wavelet transforms can reveal structural patterns not visible in the spatial domain. Multi-scale feature extraction can further improve robustness across different image resolutions. On the data augmentation side, applying more aggressive augmentation strategies, including Mixup and CutMix techniques, as well as domain-specific transformations, can improve model generalization and reduce overfitting.

7.3 Model Enhancements

Both traditional machine learning and deep learning approaches can be further enhanced. For traditional machine learning models, using ensembles of multiple classifiers, along with stacking and blending techniques, can improve prediction stability and accuracy. Automating feature engineering can also reduce manual effort and uncover more effective representations.

For deep learning models, transfer learning from large-scale datasets can provide stronger initial representations, while self-supervised pretraining can leverage unlabeled data. Few-shot learning techniques can help when labeled data is limited, and active learning strategies can guide efficient data collection by focusing on the most informative samples.

8. Conclusion

8.1 Summary of Achievements

This project successfully developed a comprehensive machine learning system for algae species classification from microscopy images. Key achievements include:

1. **Dual Approach Implementation**: Successfully implemented and compared traditional ML and deep learning approaches
2. **Feature Engineering**: Developed robust feature extraction pipeline with Fisher Discriminant Analysis for feature selection
3. **Best Performance**: Achieved 75.66% accuracy with XGBoost classifier, outperforming deep learning approach
4. **Comprehensive Evaluation**: Implemented multiple evaluation strategies including ground truth and YOLO-based evaluation
5. **Reproducible System**: Created well-structured, documented, and configurable codebase

8.2 Key Takeaways

Methodology:

1. Traditional ML with handcrafted features can outperform deep learning when data is limited
2. Feature engineering and selection are critical for traditional ML success
3. Shape features (perimeter, width, height, area) are most discriminative for algae classification
4. Class imbalance requires careful handling through weighted metrics and loss functions

References

1. Kaggle Dataset: High-throughput Algae Cell Detection (marquis03/high-throughput-algae-cell-detection)
2. YOLOv8 Documentation: Ultralytics YOLO
3. PyTorch Documentation: https://pytorch.org/docs/
4. XGBoost Documentation: https://xgboost.readthedocs.io/
5. Albumentations: https://albumentations.ai/

Appendix

A. Performance Metrics Glossary

1. **Accuracy**: (TP + TN) / (TP + TN + FP + FN)
2. **Precision**: TP / (TP + FP)
3. **Recall**: TP / (TP + FN)
4. **F1-Score**: 2 Ã— (Precision Ã— Recall) / (Precision + Recall)

Where TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

Report generated: January 2025 Project: Algae Microscopy Classifier Version: 1.0