Comparative Analysis of Oyster Seed Counting and Deploying the Pre-Trained Model on an End User System

Oyster Team

Spring 2025

Sea Grant

Louisiana Sea Grant is an organization tasked with aiding in marine and coastal life conservation, protection, and research

Sarah Bodenstein is a Postdoctoral Researcher in the oyster research lab who is working with us to achieve our project goal

Figure: Oyster Farm(First two images from left), Oyster (Next two)

Introduction

Project Goals in Spring-2025: The goal of the project is to

- Count a total amount of oyster seeds (4-6 mm) from an image of a Petri dish.
- Compare the results of our 4-6 mm model to the results of the 2-4mm models.
- Deploying the Pre-Trained Model on a system that can be used by the end user.

Introduction-Seed Collection

Collection

- Oyster seeds are collected with a bottle and then are taken out and put through filters that separate them by size
- The seeds are separated into 3 size categories: 0-2mm, 2-4mm, and 4-6mm

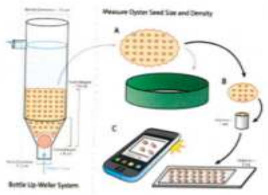


Figure: Bottle and filtering process

Annotations

Fiji and LabKit

- Our goal is to mark each individual seed in images with known seed counts. We do this to train our code to become capable of identifying each seed within an image
- To accomplish this, we used the LabKit software integrated within Fiji.

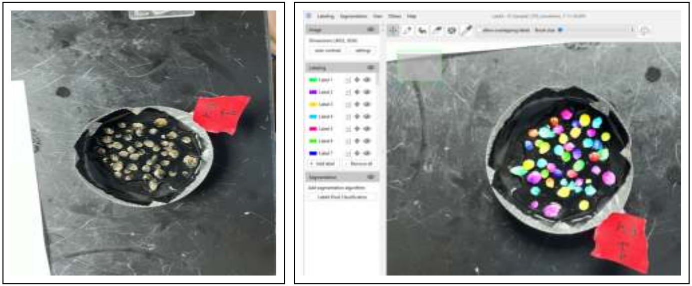


Figure: Original Oyster Image(left), Partially Annotated Image (right)

Augmentation

- **Augmentation:** In the context of machine learning and data science, augmentation refers to the process of artificially expanding a dataset by applying various transformations to the existing data. These transformations may include rotations, flips, brightness adjustments, scaling, cropping, and noise addition, among others.
- **Purpose:** The purpose of augmentation is to introduce variability in the dataset, helping models learn to generalize better and become less prone to overfitting, especially when the amount of original data is limited.

Augmentation

To mitigate issues like data scarcity and model overfitting, augmentation techniques can be classified into four categories:

- **Absent Augmentation**: No changes applied, providing a baseline comparison.
- **Basic Augmentation**: Simple transformations, including adjustments in brightness and contrast, to account for minor visual variations.
- **Intermediate Augmentation**: Adds techniques like rotation, flipping, and shifting to increase dataset diversity.
- **Advanced Augmentation**: Builds on prior methods, incorporating zoom and shear transformations to introduce further variability.

Augmentation

- **Intermediate Augmentation**: (Rotation, Flipping, and Shifting)
  - ▷ Offers a greater variety of images by introducing spatial transformations.
  - ▷ increases the model's resilience to positional changes and orientations,
  - ▷ enhancing its ability to generalize.

```python
# function for random flips/rotations of the data
def random_fliprot(img, mask):
    assert img.ndim >= mask.ndim
    axes = tuple(range(mask.ndim))
    perm = tuple(np.random.permutation(axes))
    img = img.transpose(perm + tuple(range(mask.ndim, img.ndim)))
    mask = mask.transpose(perm)
    for ax in axes:
        if np.random.rand() > 0.5:
            img = np.flip(img, axis=ax)
            mask = np.flip(mask, axis=ax)
    return img, mask

# function to give data random intensity
def random_intensity_change(img):
    img = img * np.random.uniform(0.6, 2) + np.random.uniform(-0.2, 0.2)
    return img

# this is the augmenter that will go into the training of the model
def augmenter(x, y):
    x, y = random_fliprot(x, y)
    x = random_intensity_change(x)
    sig = 0.02 * np.random.uniform(0, 1)
    x = x + sig * np.random.normal(0, 1, x.shape)
    return x, y
```

Figure: Augmentation functions

Stardist

Basics

- Uses a neural network, based on U-net, to identify star-convex polygons
- Probabilities are obtained for each pixel saying how likely a pixel is to be in a polygon
  - Probabilities found using the neural network
- 32 radial directions measured from a pixel to the boundary of the polygon
- Better at predicting cell shape than other image analysis methods

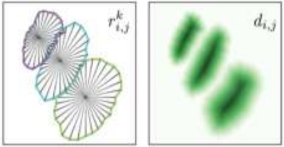


Figure: Star-Convex Polygons

Stardist

Architecture

- U-net
  - Convolutional neural network
- Post-processing
  - Non-maximal suppression

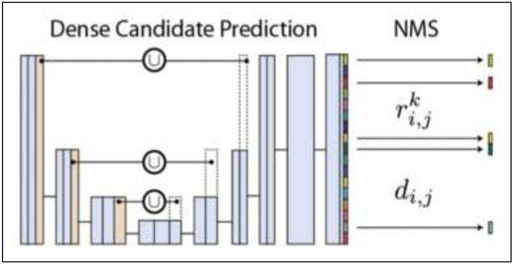


Figure: U-net architechture

Evaluation metrics

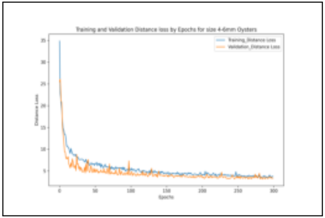
Training- Loss Functions

The loss function, a metric of the model's accuracy, being minimized:

- $L(p, p', r, r') = L_{prob}(p, p') + p' L_{dist}(p, p', r, r')$ where (p,r) are the predictions and (p',r') is the ground truth.
- $L_{prob}(p, p') = -p' ln(p) - (1 - p') log(1 - p)$
- $L_{dist}(r, r') = \frac{1}{n} \sum_k |r_k - r'_k|$

Evaluation metrics

Training- Distance Loss(15 images)



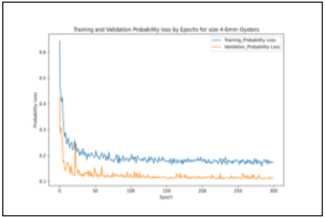
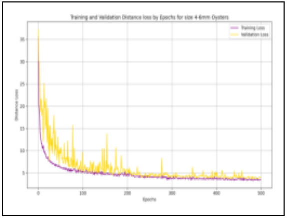


Figure: Distance Loss(left), Probability Loss(right)

- The loss statistics are based on 15 images which is half of the dataset, using an 80-20 train-test split.
- Training and validation plots are similar, and the loss flattens around 50 epochs while still improving.
- Probability loss decreases from 0.6 to 0.2 with in the first 50 epochs.
- The training and validation loss differ by about 0.05, showing consistent model performance.

Evaluation metrics

Training- Distance Loss(31 images)



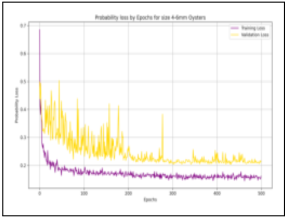


Figure: Distance Loss(left), Probability Loss(right)

- The loss statistics are based on 31 images, using an 80-20 train-test split.
- Training and validation plots are similar, and the loss flattens around 50 epochs.
- Probability loss decreases from 0.6 to 0.2 with in the first 50 epochs.
- The training and validation loss differ by about 0.05. We observed overfitting in training.

Evaluation Metrics-Results

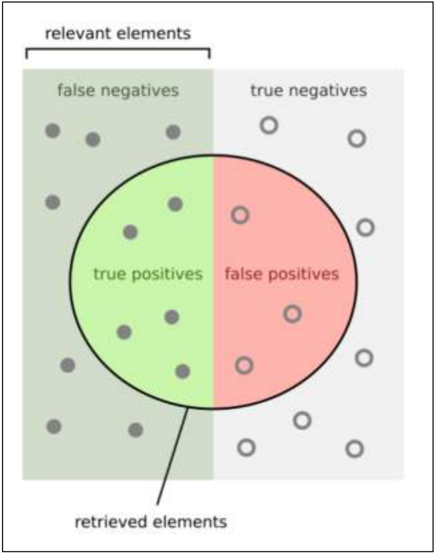


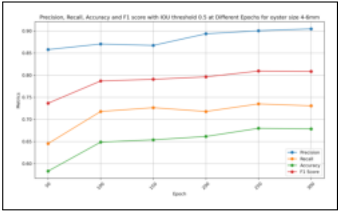
Figure: True positive, False positives

Evaluation Metrics-Tests & Results

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1 = (2 × Precision × Recall) / (Precision + Recall)
- Accuracy = TP / (TP + FP + FN)

Evaluation metrics

Results



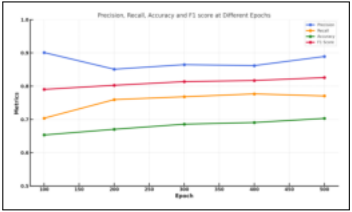


Figure: Precision, F1 score, Recall, Accuracy of two models- 15(Left), 31(Right)

- The model's validation performance improves across all metrics as training progresses in both models.
- Precision remains high throughout, proving that the models consistently make correct detections.
- Accuracy and F1 Score show a steady upward trend, meaning overall performance improves in both models.
- Left model(Validation accuracy-67%) and Right model(Validation accuracy-76%)

Accuracy rate of different sizes Oyster

- In Summer-2024, we have run the model for 2-4 mm oyster and we got more than 95 percent accuracy.
- In Fall-2024, we have run the model for 0-2 mm oyster and we got more than 81% accuracy.
- This time we run the model for 4-6 mm.

Results

- The best model gave 67% accuracy by taking half of the dataset.
- The model gave 76% accuracy with overfitting by taking the total dataset.

Model Performance

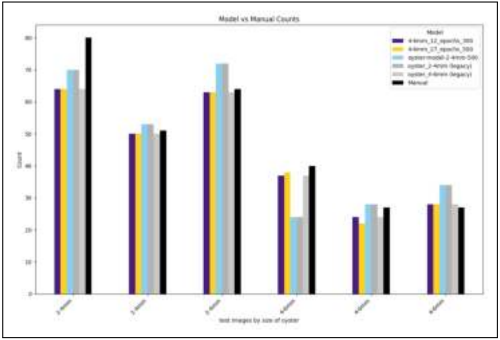


Figure: Model vs Manual counts

Model	2–4mm Acc% (MAE)	4–6mm Acc% (MAE)	4mm Acc% (MAE)
4-6mm_12_epochs_300	87.31% (10.12)	**98.48% (1.0)**	**93.34% (2.0)**
4-6mm_27_epochs_500	88.22% (9.62)	93.94% (4.0)	91.67% (2.4)
oyster-model-2-4mm-500	**93.24% (5.12)**	87.88% (8.0)	82.45% (5.8)
oyster_2-4mm (legacy)	93.24% (5.12)	87.88% (8.0)	82.45% (5.8)
oyster_4-6mm (legacy)	87.31% (10.12)	**98.48% (1.0)**	**93.34% (2.0)**

MAE = Mean Absolute Error (average absolute deviation in shell counts)

- **Accuracy Definition:** Accuracy = 100% – Mean Absolute Percentage Error (MAPE).
- **Best Models and Results:**
  2-4mm epochs 500: 93.24% accuracy, avg deviation 5.12 cells.
  4-6mm epochs 300: 98.48% accuracy, avg deviation 1.0 cell.
  4-6mm epochs 500: 93.34% accuracy, avg deviation 2.0 cells.
- **Key Finding:** Achieved approximately 98% accuracy for 4–6mm oyster counting on the test dataset.
- **Training Observation:** Training beyond 300 epochs (e.g., at 500 epochs) led to reduced performance due to overfitting.

GUI

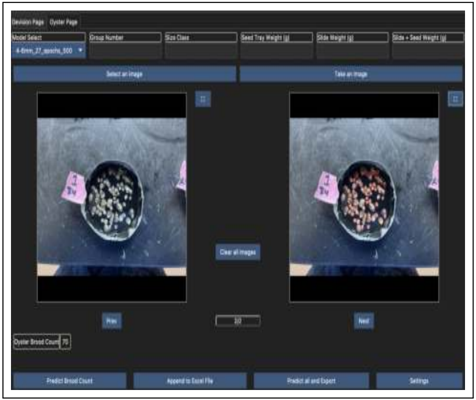


Figure: GUI Prediction of Oyster Image(4-6mm)

Deploying to End User Device

different hardware options we considered to deploy our pre-trained modals and GUI

- ✓Designated Raspberry Pi Device
- Local deployment to hand held devices
- Deployment of a client-server solution

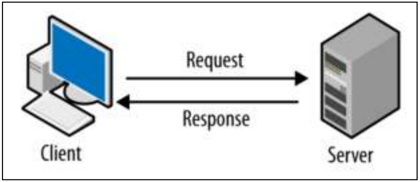


Figure: client-server

Deploying to End User Device

We landed on the use of the Raspberry Pi for a few reasons:

- Compatibility with our software
- Widely available hardware and replacement parts
- Small form factor and “relatively” easy to make it into a portable device

Figure: Raspberry Pi Implementation

Deploying to End User Device

We worked with the GUI team to get the program running on the Raspberry Pi

- Updated the code to improve responsiveness
- Implemented new features to allow for ease of use
- Testing of software on Raspberry Pi platform

Additional Work

- designed and 3d-printed a housing for the Raspberry Pi
- Made a video tutorial for assembly of the Raspberry Pi
- Made a single file binary distribution for the Raspberry Pi
- bench-marked model performance on the Raspberry Pi

Figure: Raspberry Pi Implementation

In the Future

- We aim to update and fix any errors still in the GUI.
- We aim to write comprehensive documentation for other people to follow our work.
- We aim to update the 3d models fixing issues we encountered while working.
- We intend to write an article based on our research on the Oyster project.

Figure: 4-6 mm Oysters

Acknowledgments

We would like to thank

- Dr. Peter Wolenski for guiding and supporting us.
- Dr. Nadejda Drenska for guiding and supporting us.
- Gowri Priya Sunkara, Dow Draper and Shalini Shalini for their assistance in the machine learning project.
- The Department of Mathematics for foreseeing the utility of "Chaos" and the subsequent purchase.
- Elizabeth M. Robinson,Director, Louisiana Sea Grant Research Lab and Michael C. Viosin Oyster Hatchery and Dr. Sarah Bodenstein, Post Doc for giving us an opportunity to apply the Machine learning Algorithm on the Oysters project.
- Nikkos Svoboda, Computer Analyst for introducing and making us familiar with the workings of the high-performance computing system "Chaos".

Acknowledgements

Figure: Sarah Bodenstein(Up), Previous Oyster Teams (Below)

References

References:

- Schmidt, U., Weigert, M., Broaddus, C., Myers, G. (2018). Cell Detection with Star-Convex Polygons. Medical Image Computing and Computer Assisted Intervention – MICCAI 2018.
- https://nbviewer.org/github/stardist/stardist/blob/master/examples/other2D/multiclass.ipynb.
- https://github.com/LSU-Devision/Summer-2024
- https://analyticalscience.wiley.com/content/article-do/stardist-application-deep-learning-tool-phase-contrast-cell-images.
- https://www.laseagrant.org/outreach/aquaculture/oyster-research-lab/lab-staff/about/
- https://www.sarahbodenstein.com/
- https://github.com/LSU-Devision/GUI
- https://github.com/LSU-Devision/Louisiana-Sea-Grant.

Team Members

Department of Mathematics Advisors:

- Dr. Peter Wolenski
- Dr. Nadejda Drenska

Team Members:

- Sanjida Nupur
- Mark C Herpin
- Maxwell T Shroyer

Thank you for listening!