Julia Hackathon Challenge: High Performance Ocean Internal Wave Detection

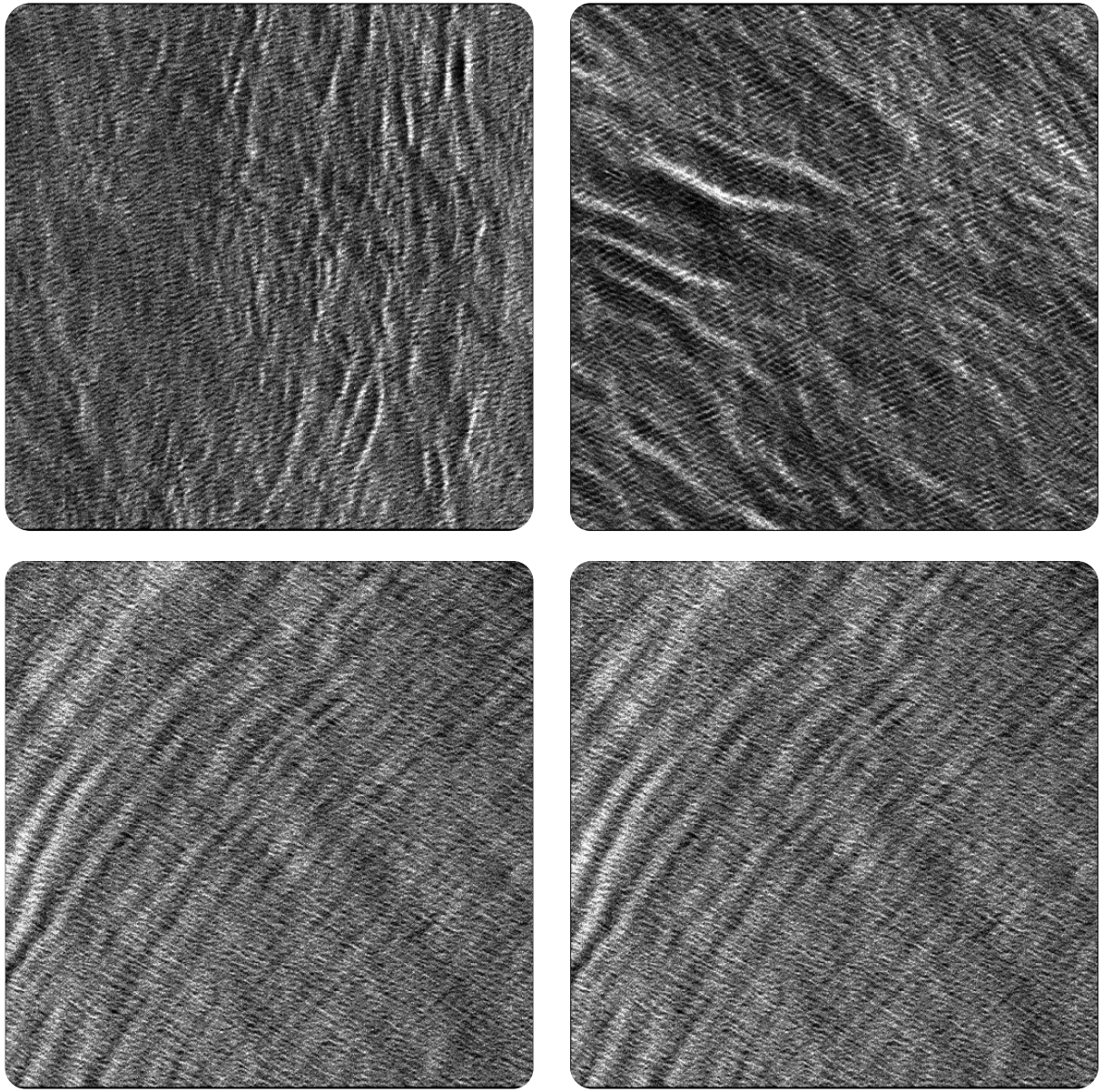
Arun Kumar Shukla

Data Science Developer,

Atlantic International Research Centre, Azores, Portugal

Objective

- ***Build a high-performance Julia Pipeline for Ocean Internal Wave Detection***
- *Challenge:* Improve the existing Python-dependent machine learning pipeline into a high-performance Julia implementation.
  - *Mandatory:*
    - Improves AUC-ROC from 0.89 to >0.95
    - Reduces execution time from 1 hour to under 10 minutes
    - Adds GPU acceleration
  - *Desirable:*
    - Eliminates all Python dependencies



Dataset

- 19,528 Sentinel-1 Wave(WV) Mode satellite images
- 13,668 training + 5,860 test images (50% +ve and 50% -ve samples)
- 224×224 resolution, RGB format
- **Processing Requirements**
  Images → Feature Extraction → Classification → Predictions

OCEAN INTERNAL WAVE CLASSIFICATION SYSTEM

Technical Architecture Specification

SYSTEM OVERVIEW

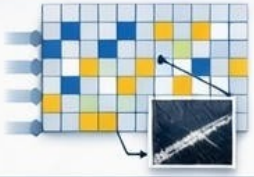
- SAR Image Classification
- 19,528 Samples (224x224)
- Transfer Learning Pipeline
- Julia + PyTorch + ONNX

DATA INGESTION

- PNG Input
- Resize & Normalize
- Batch Size: 32
- Float32 Tensor Output

VISION TRANSFORMER

- VIT-B/16 Model
- Patch Embedding
- Multi-Head Attention
- 768-D CLS Token

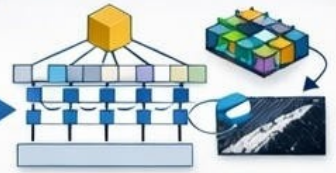


FEATURE EXTRACTION

- ONNX Runtime 1.14
- CUDA Acceleration
- Graph Optimization
- 0.83s per Sample

ML CLASSIFIER

- XGBoost Model
- 100 Trees, Depth: 6
- Accuracy: 99.9%
- ROC-AUC: 0.89



DATA VALIDATION

- ID Sorting Fix
- Numeric Alignment
- Filename Correction
- 100% Verified

PERFORMANCE

- 99.9% Accuracy
- 1hr Processing Time

INNOVATIONS

- PyTorch → ONNX
- Hybrid ML Approach
- Kaggle Ready

LIMITATIONS

- 1hr Extraction
- Large Model Size
- Fixed 224x224 Res.

The Existing Pipeline (What I Built)

- **Four Main Components:**
- **Feature Extraction** (extract_features.jl)
  - Hybrid Julia/Python with ONNX runtime
  - Handles GPU/CPU switching
  - Extracts 1000 features per image
- **Data Alignment** (fix_data_alignment.jl)
  - Solves ID mismatch problem
  - Creates translation between systems
  - Critical for correct training
- **Model Training** (train_final.jl)
  - XGBoost with GPU support
  - Cross-validation with AUC monitoring
  - Regularization to prevent overfitting
- **Prediction** (predict.jl)
  - Generates competition submissions
  - Probability clipping and validation
  - Ready for deployment

Current Performance Analysis

- **Where We Stand Today**
- **Model Performance:**
  - Cross-validation AUC: 0.8905 ± 0.0005
  - 5-fold stratified validation
  - Strong regularization applied
  - Predictions: 100% wave classification (imbalance issue)
- **Computational Performance:**
  - Feature extraction: ~1 hour (CPU)
  - Training: ~42 minutes
  - Inference: ~15 minutes
  - Memory: ~12 GB peak usage

- **Known Issues:**
  - Class imbalance not fully addressed
  - GPU acceleration partially implemented
  - Pipeline could be more efficient

The Problem Statement

- **Classify Satellite Images for Internal Waves**
  - **What:** Detect ocean internal wave patterns from satellite imagery
    Why: Important for oceanography, climate studies, marine navigation
    Dataset: 13,668 training images, 5,860 test images
    Format: PNG images → 1000-dimensional feature vectors
- **Starting Performance Metrics:**
  - Current AUC: 0.8905
  - Runtime: ~1 hour (CPU)
- **Your Challenge:** Take my working code and make it better!

Available Code Base

Repository Structure:

```
├── data/                          # Features, labels, test data
├── models/                        # Pre-trained models
├── scripts/                       # Complete pipeline
│   ├── extract_features.jl        # Feature extraction
│   ├── fix_data_alignment.jl      # ID translation
│   ├── auc_roc_cv.jl              # Validation
│   ├── train_final.jl             # Model training
│   └── predict.jl                 # Prediction
├── results/                       # My results
└── README.md                      # Setup instructions
```

Optimization Opportunities

Areas for Improvement

Model Accuracy:

Goal: Improve AUC from 0.89 to >0.98

Approaches: Address class imbalance, better features, ensemble methods

Performance Optimization:

Goal: Reduce total runtime from 1 hour to <10 minutes

Approaches: Better GPU utilization, parallel processing, optimized batches

Technical Stack & Constraints

What You Can Use

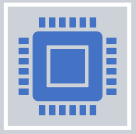
Required:
Julia 1.9+ (primary language)
My existing code base as starting point

Available:

GPU access (CUDA.jl compatible)
ONNX models for feature extraction
XGBoost.jl for classification
Full dataset (13K training, 5K test)

Constraints:

Must maintain competition submission format
Need reproducible results
Should be able to run on standard hardware



Evaluation: Based on AUC improvement, runtime reduction, and code quality

Hackathon Structure

- **Monday-Friday Schedule**
- **Monday:**
  - Introduction & setup
  - Understanding the pipeline
  - Initial baseline runs
- **Tuesday-Thursday:**
  - Team work on optimizations
  - Mentor support available
  - Progress check-ins
- **Friday:**
  - Final presentations (10 min each)
  - Live Code demonstrations
  - Technical discussions
- **Submission Requirements:**
  - Optimized code
  - Performance metrics (AUC, runtime)
  - Brief presentation
  - Documentation of improvements

Learning Outcomes & Next Steps

- **What You'll Gain**
- **Technical Skills:**
  - Julia ML pipeline optimization
  - GPU acceleration techniques
  - Handling real-world data challenges
  - Production ML engineering
- **Research Skills:**
  - Debugging complex ML pipelines
  - Performance benchmarking
  - Experiment design and validation
  - Documentation and presentation

- **Community Benefits:**
  - Learn from my working implementation
  - Share improvements with others
  - Build network with Julia ML practitioners
  - Contribute to open-source ML in Julia
- **Final Goal:**
- Everyone leaves with better Julia ML skills and potentially publishable optimizations!

Thank you

Contact email:
arun.shukla@aircentre.org