NTNU

elsys
ELEKTRONISK SYSTEMDESIGN & INNOVASJON

Final Report

Project: Deep Learning for Segmentation of Hyperspectral Satellite Images

Student: Arya Raeesi

Supervisor: Roger Birkeland

Contact person at the problem owner: Roger Birkeland

Date: 2025-04-27

Contents

1 Problem Statement

The problem addressed in this project concerns the adaptation of an existing convolutional neural network, the 1D-JustoLiuNet, from the HYPSO-1 satellite to the HYPSO-2 satellite at the SmallSat Lab at NTNU. The network was originally trained on data from HYPSO-1 and now needs to be adjusted and retrained on new hyperspectral images from HYPSO-2 to support continued onboard use during operational deployment.

Hyperspectral satellite images are datasets that capture information across a large number of narrow wavelength bands, providing a rich spectral representation for each pixel in the image. This spectral information enables accurate classification of surface types such as sea, land, and cloud, based on their unique spectral signatures.

To automate this classification, a convolutional neural network (CNN) is employed, a type of machine learning model particularly well suited for image analysis. The goal of the project is to train and evaluate this network using the new HYPSO-2 images, and to assess the model's ability to generalize to diverse and challenging imaging conditions.

2 Solution

2.1 Hyperspectral Images

Hyperspectral images are images that capture information across contiguous spectral bands, typically ranging from the visible to the near-infrared spectrum. Unlike traditional RGB images that contain only three channels (red, green and blue), hyperspectral data provides a full reflectance spectrum for each pixel in the image, enabling fine-grained discrimination between materials based on their spectral signatures. This spectral information makes hyperspectral imaging particularly useful for our task of segmenting sea, land and clouds from HYPSO-2 satellite images. Figure 1 shows an illustration of a satellite with a hyperspectral camera.

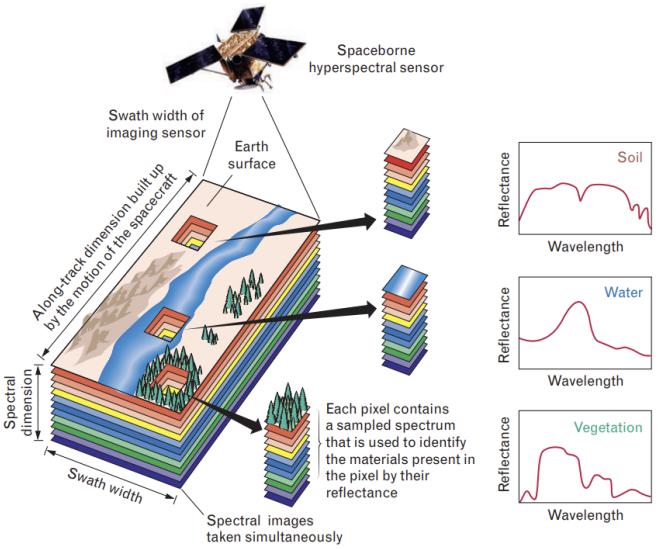


Figure 1: Illustration of a satellite with a hyperspectral camera [1].

2.2 Neural Networks

Neural networks are computational models inspired by the structure of the human brain, consisting of interconnected layers of artificial neurons [2, page 15-16]. Each neuron applies a mathematical function to its input and passes the result to the next layer. These networks are trained using large datasets and learn to recognize patterns, classify data, or make predictions [2, page 15-16].

2.2.1 Parts of a Neural Network

A neural network consists of three main components: an input layer, one or more hidden layers, and an output layer, as illustrated in Figure 2.

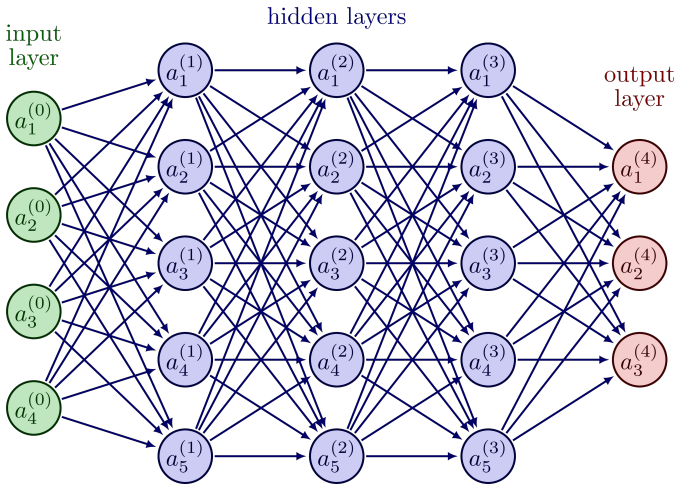


Figure 2: Example of a neural network [3].

Each neuron in the network receives input values x_i, each associated with a corresponding weight w_i, which determines the relative importance of that input. Additionally, a bias term b is added to the weighted sum, allowing the neuron to adjust its output independently of the input values. The resulting value z is then passed through an activation function ϕ, which introduces non-linearity and enables the network to model complex relationships. Mathematically, this process can be described in Equation 1 [2, page 168-172]:

$$z = \sum_{i=1}^{n} w_i x_i + b \implies \text{output} = \phi(z) \tag{1}$$

Together, these components allow neural networks to model data patterns and solve machine learning tasks.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network designed for analyzing data with spatial or temporal structure, such as images or time series [2, page 254]. Unlike fully connected networks, CNNs apply convolutional filters that move across the input data, allowing the network to detect local patterns and spatial hierarchies [2, page 330]. Figure 3 illustrates a 2D convolution operation.

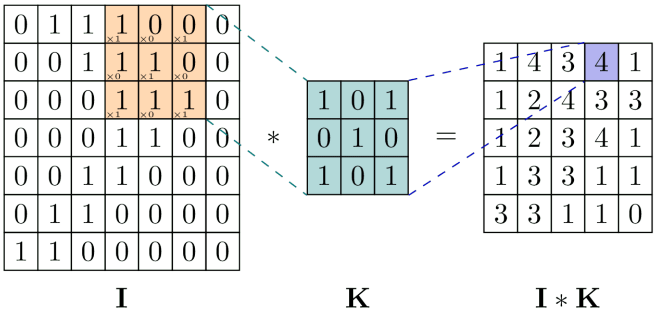


Figure 3: Example of a 2D convolution operation. A small filter (K) slides across the input matrix (I), computing a weighted sum at each location. This produces a feature map (I * K) where local patterns, such as edges or textures, are highlighted. Convolution enables the network to extract spatial features while keeping the number of parameters manageable [4].

CNNs are especially well-suited for hyperspectral images because these images contain both spatial and spectral dimensions. In our case, the 1D-JustoLiuNet is used to process the spectral signature of each pixel independently, focusing solely on the spectral axis.

2.3 1D-JustoLiuNet

1D-JustoLiuNet is a lightweight one-dimensional convolutional neural network designed for processing hyperspectral image data [5, page 3-4]. The network consists of four sequential convolutional layers, each followed by a ReLU activation function. The ReLU operation is illustrated in algorithm 1 [2, page 175].

Algorithm 1: ReLU Activation Function

Input: Feature vector $x = [x_1, x_2, \ldots, x_n]$
Output: Activated vector $y = [y_1, y_2, \ldots, y_n]$

for $i \leftarrow 1$ **to** n **do**
  $y_i \leftarrow \max(0, x_i)$;
return y

After the ReLU activation, a 1D MaxPooling operation is initialized to reduce the dimensionality of the feature maps while preserving important spectral features. This operation is illustrated in algorithm 2 [2, page 339-342].

Algorithm 2: 1D MaxPooling

Input: Feature vector $x = [x_1, x_2, \ldots, x_n]$, pool size k
Output: Pooled vector y

Initialize $y \leftarrow []$;
for $i \leftarrow 0$ **to** $n - k$ k **do**
  $window \leftarrow [x_i, x_{i+1}, \ldots, x_{i+k-1}]$;
  $max \leftarrow \max(window)$;
  Append max to y;
return y

After the convolutional stages, the feature maps are flattened into a single vector, by

flattening it. The flattening algorithm is illustrated in algorithm 3 [2, page 207].

Algorithm 3: Flattening a 3D Tensor

Input: 3D tensor T with shape (B, C, L)
Output: 2D matrix M with shape $(B, C \cdot L)$

for *each sample* T_b *in batch* **do**
    $M_b \leftarrow \text{Flatten}(T_b)$;
    Add M_b as row in M;
return M

Lastly, the vector is passed through a fully connected layer that outputs the final class predictions [2, page 203]. This algorithm is illustrated in algorithm 4 [2, page 203].

Algorithm 4: Fully Connected Layer

Input: Vector $x = [x_1, ..., x_m]$, weight matrix W of shape $(n \times m)$, bias vector b of length n
Output: Output vector $y = [y_1, ..., y_n]$

for $i \leftarrow 1$ **to** n **do**
    $y_i \leftarrow W_i \cdot x + b_i$;
return y

With all the algorithms now defined, the forward pass of the 1D-JustoLiuNet can be defined in algorithm 5 [5, page 11].

Algorithm 5: Forward pass of the 1D-JustoLiuNet

Input: Input spectral vector $x \in \mathbb{R}^{1 \times N}$
Output: Predicted class probabilities $y \in \mathbb{R}^{C}$

$x \leftarrow \text{Conv1d}(x, \text{kernel}_1)$; // First convolution layer
$x \leftarrow \text{ReLU}(x)$
$x \leftarrow \text{MaxPool1d}(x)$
$x \leftarrow \text{Conv1d}(x, \text{kernel}_2)$; // Second convolution layer
$x \leftarrow \text{ReLU}(x)$
$x \leftarrow \text{MaxPool1d}(x)$
$x \leftarrow \text{Conv1d}(x, \text{kernel}_3)$; // Third convolution layer
$x \leftarrow \text{ReLU}(x)$
$x \leftarrow \text{MaxPool1d}(x)$
$x \leftarrow \text{Conv1d}(x, \text{kernel}_4)$; // Fourth convolution layer
$x \leftarrow \text{ReLU}(x)$
$x \leftarrow \text{MaxPool1d}(x)$
$x \leftarrow \text{Flatten}(x)$
$y \leftarrow \text{Linear}(x)$; // Fully connected layer
return y

2.4 Preprocessing

Before feeding the hyperspectral input to the neural network, the raw spectral values are normalized to ensure numerical stability and to bring all input features onto a common scale. In this project, min-max normalization is applied to each pixel. Equation 2 defines the min-max normalization.

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min} + \epsilon}, \quad \epsilon \ll 1 \tag{2}$$

The normalization happens band-by-band, meaning that for instance band 29 in pixel X is normalized for all band 29 in the whole image. This transformation maps the values to a range close to $[0, 1]$. To avoid division by zero, a small constant $\epsilon \ll 1$ is added in the denominator.

2.5 Dataset and Format

The dataset used in this project consists of raw hyperspectral images stored in Band Interleaved by Pixel (BIP) format. In this structure, the complete spectral profile of each pixel is stored sequentially, which is ideal for spectral analysis on a per-pixel basis. These `.bip` files are preprocessed and loaded into memory as tensors for input to the neural network.

In addition to the raw data, label data is stored in `.dat` files, where each entry corresponds to a pixel-wise class annotation (sea, land or cloud). These `.dat` files were created semi-automatically using ENVI. The labels are used in training to guide the network in learning class-specific spectral patterns.

2.6 Model Training Setup

The training pipeline follows the sequence shown in Figure 4.

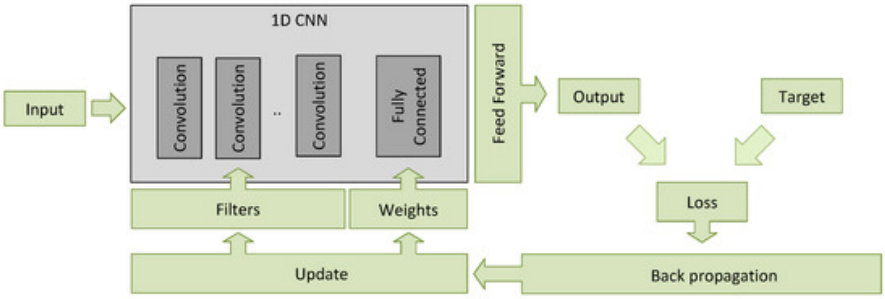


Figure 4: The training pipeline utilized in this project [6].

As illustrated in Figure 4, the CNN gets an input, and uses that input, compares with the target and updated the filters and weights of the trained CNN. By saving the filters and weights, inference can be done with the CNN. Inference is the process of the CNN classifying an image based on its trained model. In simpler terms, good filters and weights are the parameters needed for the 1D-JustoLiuNet in the HYPSO-2 satellite.

The training pipeline relies on several key hyperparameters that control how the model learns. Batch size defines how many training samples are processed before the model updates its internal parameters. It affects memory usage and the stability of gradient estimates. Learning rate determines how much the model's weights are adjusted in response to the estimated error each time the weights are updated. It plays an important role in convergence speed and stability. Label smoothing is a regularization technique that slightly adjusts the target labels, preventing the model from becoming overly confident in its predictions and improving generalization. Epochs refer to the number of complete passes through the entire training dataset during training, allowing the model to iteratively refine its performance. Table 1 has these variables and its values for this project.

Table 1: Variables used in training pipeline and its values.

Variable	Value
Batch size	128
Learning rate	0.001
Label smoothing	0.1
Epochs	10

In addition, the pipeline uses `AdamW` as the optimizer, `StepLR` as the scheduler, and `CrossEntropyLoss` as the loss function. Section 2.6.1, 2.6.2 and 2.6.3 explain these parts of the pipeline.

2.6.1 Optimizer - `AdamW`

The optimizer is responsible for updating the model's parameters during training to minimize the loss function. `AdamW` is a variant of the `Adam` optimizer that decouples weight decay from the gradient update, improving generalization performance. It's needed because without an optimizer, the network cannot learn. `AdamW` was chosen because it performs well on sparse gradients and allows for weight decay regularization. Algorithm 6 illustrates the `AdamW` optimizer [7].

Algorithm 6: AdamW Optimizer

Input: Gradient g_t, parameters w_t, learning rate α, momenta β_1, β_2, weight decay λ

Output: Updated weights w_{t+1}

$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

$\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$

$w_{t+1} \leftarrow w_t - \alpha(\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda w_t)$

2.6.2 Scheduler - `StepLR`

A learning rate scheduler dynamically adjusts the learning rate during training to help the model converge better. `StepLR` reduces the learning rate by a fixed factor every

few epochs. This was beneficial to the pipeline because using a static learning rate can cause overshooting or slow convergence. `StepLR` helps refine the training as it progresses. Algorithm 7 shows how the `StepLR` scheduler works [7].

Algorithm 7: StepLR Scheduler

Input: Initial learning rate α_0, step size s, gamma γ, epoch t
Output: Learning rate α_t

if $t \ \% \ s == 0$ **then**
  $\alpha_t \leftarrow \alpha_0 \cdot \gamma^{\lfloor t/s \rfloor}$
else
  $\alpha_t \leftarrow \alpha_{t-1}$

2.6.3 Loss Function - `CrossEntropyLoss`

The loss function measures how far the model's predictions are from the true labels. `CrossEntropyLoss` is commonly used for multi-class classification problems. It's needed because the loss function drives the learning. `CrossEntropyLoss` is ideal for classification since it heavily penalizes incorrect confident predictions. Algorithm 8 shows how `CrossEntropyLoss` works [7].

Algorithm 8: CrossEntropyLoss

Input: Predicted vector $p = [p_1, ..., p_C]$, true class label y
Output: Loss $\mathcal{L}$

$\mathcal{L} \leftarrow -\log(p_y)$

2.7 Classification Metrics

To evaluate the model's performance, several standard classification metrics are used. These metrics quantify different aspects of the prediction quality and are based on the concepts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

2.7.1 Accuracy

Accuracy measures the overall proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

It indicates how often the model correctly predicts the class compared to the total number of predictions.

2.7.2 Precision

Precision measures how many of the predicted positive instances are actually correct:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

High precision means that when the model predicts a certain class, it is usually correct.

2.7.3 Recall

Recall measures how many of the actual positive instances were correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

High recall means the model successfully finds most instances of the class.

2.7.4 F1-Score

The F1-score is the harmonic mean of precision and recall:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

It balances precision and recall, especially useful when there is class imbalance.

2.7.5 Macro Average

Macro averaging computes the metric independently for each class and then takes the unweighted mean:

$$\text{Macro Avg} = \frac{1}{N} \sum_{i=1}^{N} \text{Metric}_i \tag{7}$$

where N is the number of classes. Each class contributes equally, regardless of its support.

2.7.6 Weighted Average

Weighted averaging computes the metric for each class and weights it by the number of instances (support) in each class:

$$\text{Weighted Avg} = \frac{\sum_{i=1}^{N} \text{Support}_i \times \text{Metric}_i}{\sum_{i=1}^{N} \text{Support}_i} \tag{8}$$

This ensures that classes with more instances have a greater impact on the final score.

2.7.7 Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model. It compares the predicted labels against the true labels and is structured as follows:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

2.8 Evaluation Methodology

To evaluate the performance of the model, the dataset was selected to include both relatively easy and more challenging images. Easy images are those that share similar characteristics with the training data. For instance, images from the same general geographic region. These help assess how well the model performs under familiar conditions.

Challenging images, on the other hand, include examples that differ more noticeably from the rest. This could involve, for example, spectral data from different climates or terrain types. Such variation introduces a moderate level of difficulty and helps test the model's ability to generalize beyond the conditions it was trained on.

This approach ensures that the evaluation captures both the model's learning effectiveness and its robustness, without being overly dependent on similarity to the training data.

2.9 GPU Acceleration

GPUs (Graphics Processing Units) are essential for accelerating deep learning workloads due to their massively parallel architecture, allowing thousands of operations to be executed concurrently [8]. In this project, GPU acceleration was leveraged to reduce training time for the 1D-CNN model by parallelizing matrix operations such as convolutions, activations, and gradient computations during backpropagation.

The model was trained using the NVIDIA GeForce RTX 3080 GPU with CUDA (Compute Unified Device Architecture), a parallel computing platform and API developed by NVIDIA. CUDA allows developers to write code that runs directly on the GPU, providing access to hundreds or thousands of GPU cores for simultaneous computation. In frameworks like PyTorch, CUDA handles tensor operations by distributing the work across multiple threads, optimizing both memory access and computational throughput [8].

CUDA-enabled training not only speeds up the forward and backward passes of the CNN, but also ensures efficient utilization of GPU memory and compute resources. This is particularly important for hyperspectral data, which involves high-dimensional inputs that benefit greatly from parallel processing.

2.10 Code Implementation Diagrams

After presenting the components and flow of the system, it is useful to include diagrams that provides an overview of the code structure. Firstly, a component diagram was created to illustrate the main functional parts of the system and how they are interconnected. This diagram highlights the overall structure of the codebase, showing the key modules responsible for data management, model training, evaluation, and utility functions. The diagram is provided in Figure 5.

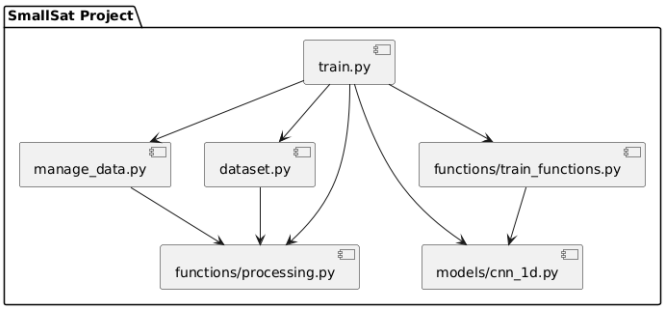


Figure 5: Component diagram of code.

Secondly, a sequence diagram was developed to describe the typical runtime behavior of the system during model training. This diagram captures the flow of operations between the main classes and functions, from data loading to training loop iterations and evaluation steps. It shows how control moves dynamically through the system during execution. The diagram is provided in Figure 6.

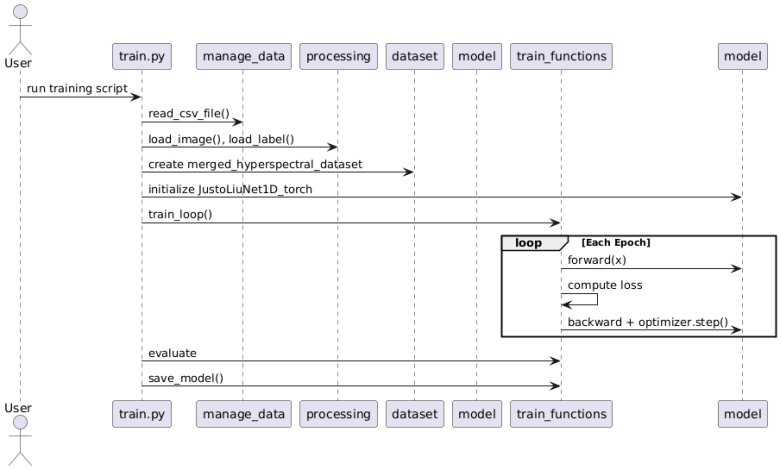


Figure 6: Sequence diagram of code.

Thereafter, an activity diagram was designed to present the logical workflow of the entire training and evaluation process. It focuses on the decision points, data transformations, and actions performed throughout the pipeline. The activity diagram complements the previous diagrams by providing a detailed view of the operational logic and conditional flows in the project. The diagram is provided in Figure 7.

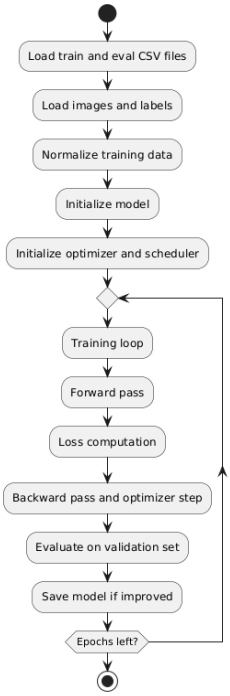


Figure 7: Activity diagram of code.

Lastly, the class diagram summarizes how the different parts of the codebase are organized and how they interact with each other. Figure 8 shows the structure of the implemented codebase.

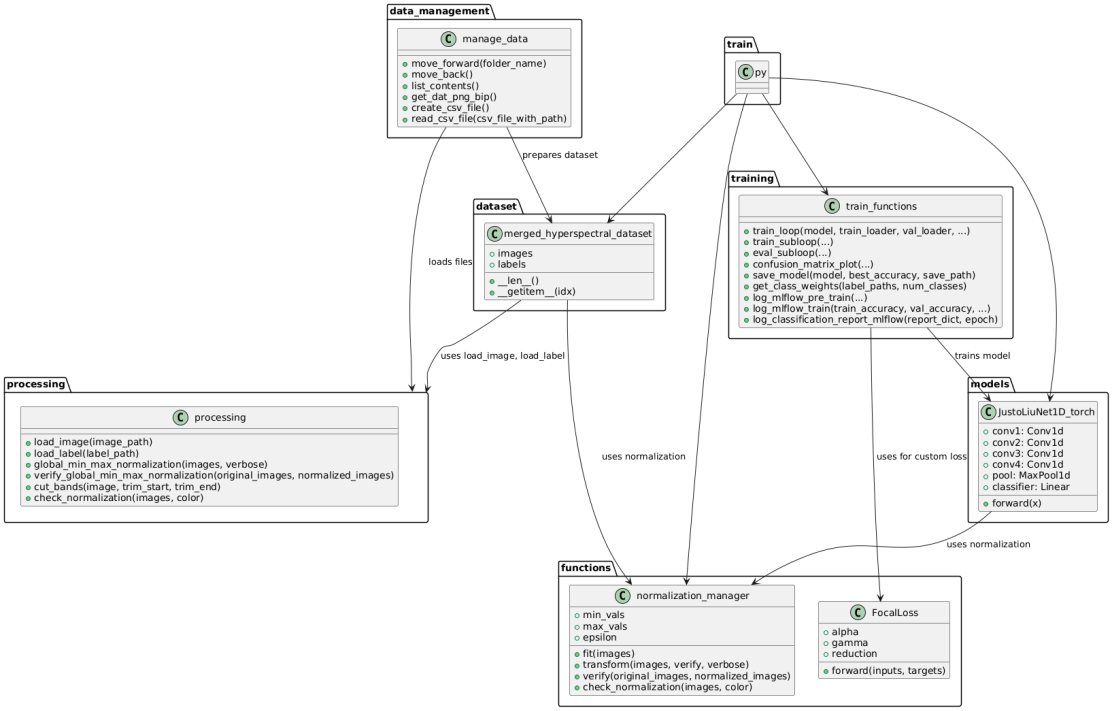


Figure 8: Class diagram of code. Note that some elements have been shortened to keep the diagram understandable (see Figure 22 for complete image). In addition, `FocalLoss` has been implemented in the code, but isn't used, that's why it's present here, but isn't noted in Section 2.6.

The code itself can be found in appendix A.

3 Verification and Test

3.1 Tests

Table 2 outlines the tests presented in this report. There are several variables set in Table 2, their meanings are:

1. **Sanity test:** A basic check to verify that the model and data pipeline function correctly.
2. **Similarity test:** A test where the evaluation images are similar to the training data, assessing how well the model generalizes under low variation.
3. **Diversity test:** A test where both the training and evaluation sets are sampled from a pool of highly varied images, meaning the images within and between the sets have low visual similarity, testing the model's ability to generalize across diverse inputs.
4. **Difficulty level:** An indicator of how hard a test is, based on the similarity between training and evaluation data.
5. **Similarity index:** A measure indicating whether the images within the dataset are visually similar to each other, where 1 means high internal similarity and 0 means high diversity.

Table 2: Overview of defined tests, difficulty levels, and similarity indices.

Test ID	Test type	Difficulty level	Images in test	Similarity index
T01	Sanity test	Easy	6	1
T02	Sanity test	Easy	8	0
T03	Sanity test	Easy	25[1]	1
T04	Similarity test	Medium	9	1
T05	Diversity test	Hard	12	0
T06	Diversity test	Hard	27	0

The results from the tests in Table 2 are presented in the subsections of Section 3.1. In addition, Appendix C contains the images chosen for each test.

3.1.1 T01

The train- and evaluation accuracy per epoch is plotted in Figure 9.

[1]There's 1 image for both training and evaluating, but during training the image creates 24 copies of itself. In simpler terms, the model gets trained on 25 exact copies of one image, and gets evaluated on that same image.

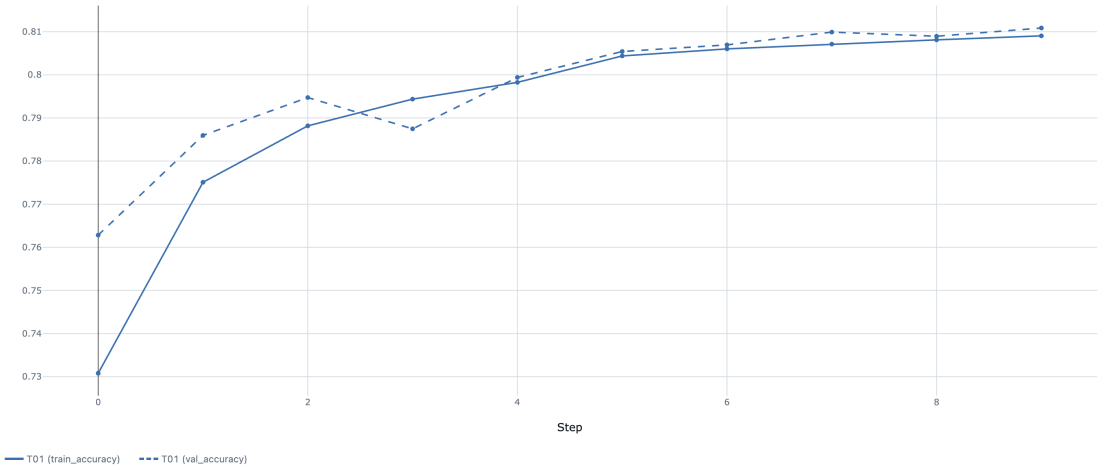


Figure 9: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T01.

In addition, the classification report from the test's final epoch has been illustrated in Table 3.

Table 3: Classification report from the final epoch. Test ID: T01.

Class	Precision	Recall	F1-Score	Support
Cloud	0.61	0.01	0.01	128413
Land	0.85	0.89	0.87	2562136
Sea	0.74	0.74	0.74	1227547
Accuracy			0.81	3918096
Macro Avg	0.73	0.54	0.54	3918096
Weighted Avg	0.80	0.81	0.80	3918096

Lastly, the last epoch's confusion matrix has been illustrated in Figure 10.

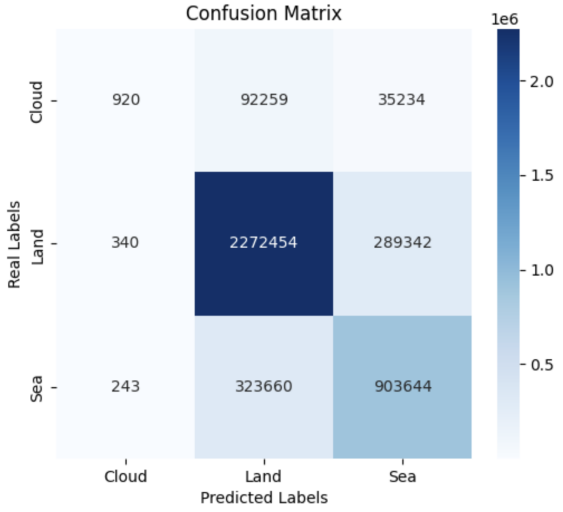


Figure 10: Confusion matrix from the final epoch. Test ID: T01.

3.1.2 T02

The train- and evaluation accuracy per epoch is plotted in Figure 11.

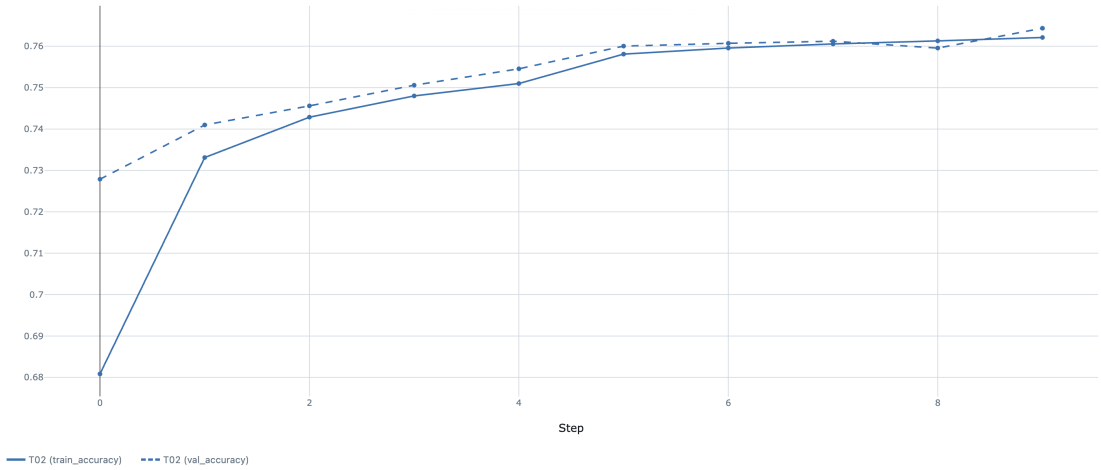


Figure 11: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T02.

In addition, the classification report from the test's final epoch has been illustrated in Table 4.

Table 4: Classification report from the final epoch. Test ID: T02.

Class	Precision	Recall	F1-Score	Support
Cloud	0.73	0.73	0.73	1019113
Land	0.82	0.79	0.81	2687515
Sea	0.69	0.73	0.71	1517500
Accuracy			0.76	5224128
Macro Avg	0.75	0.75	0.75	5224128
Weighted Avg	0.77	0.76	0.77	5224128

Lastly, the last epoch's confusion matrix has been illustrated in Figure 12.

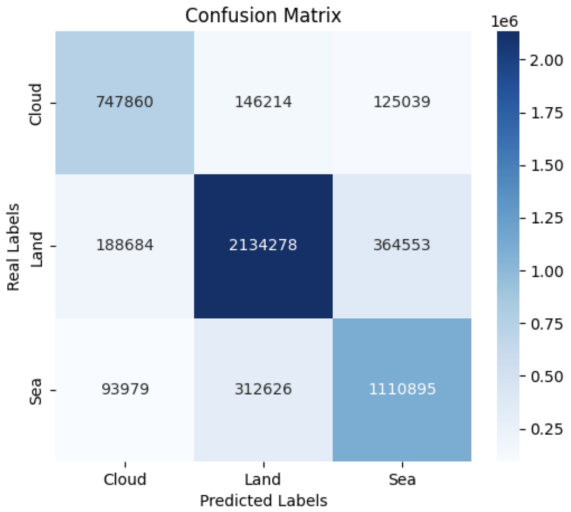


Figure 12: Confusion matrix from the final epoch. Test ID: T02.

3.1.3 T03

The train- and evaluation accuracy per epoch is plotted in Figure 13.

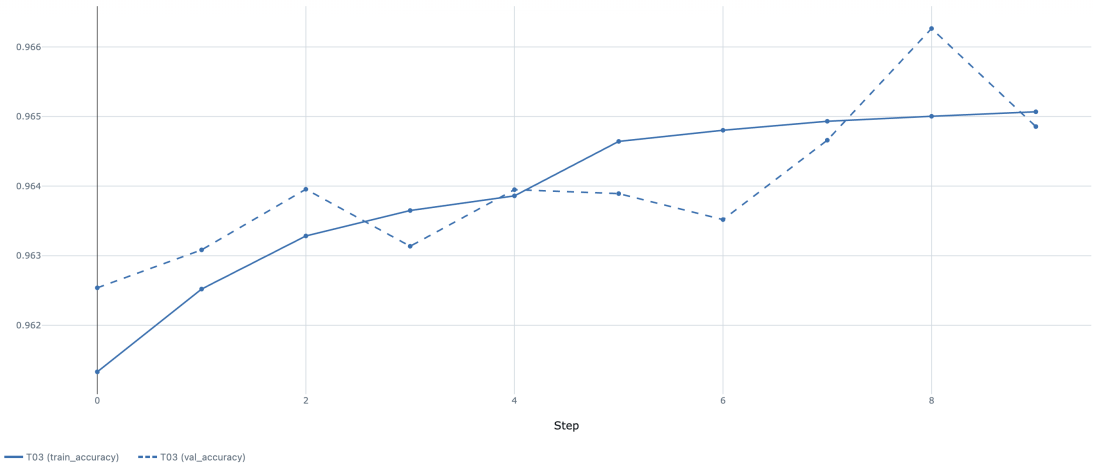


Figure 13: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T03.

In addition, the classification report from the test's final epoch has been illustrated in Table 5.

Table 5: Classification report from the final epoch. Test ID: T03.

Class	Precision	Recall	F1-Score	Support
Cloud	0.00	0.00	0.00	152
Land	0.97	1.00	0.98	627512
Sea	0.82	0.13	0.22	25352
Accuracy			0.96	653016
Macro Avg	0.59	0.38	0.40	653016
Weighted Avg	0.96	0.96	0.95	653016

Lastly, the last epoch's confusion matrix has been illustrated in Figure 14.

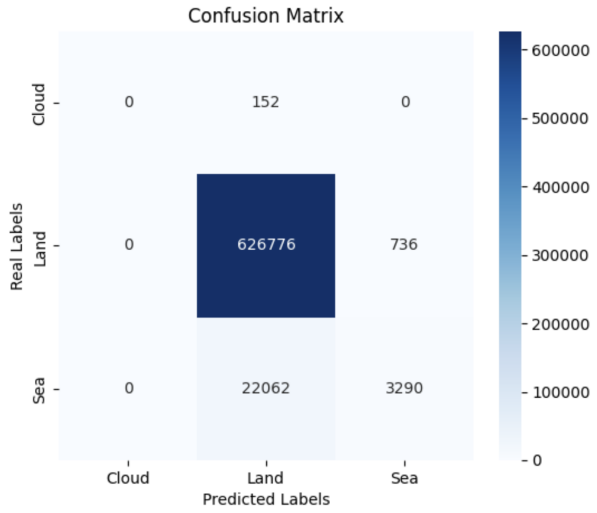


Figure 14: Confusion matrix from the final epoch. Test ID: T03.

3.1.4 T04

The train- and evaluation accuracy per epoch is plotted in Figure 15.

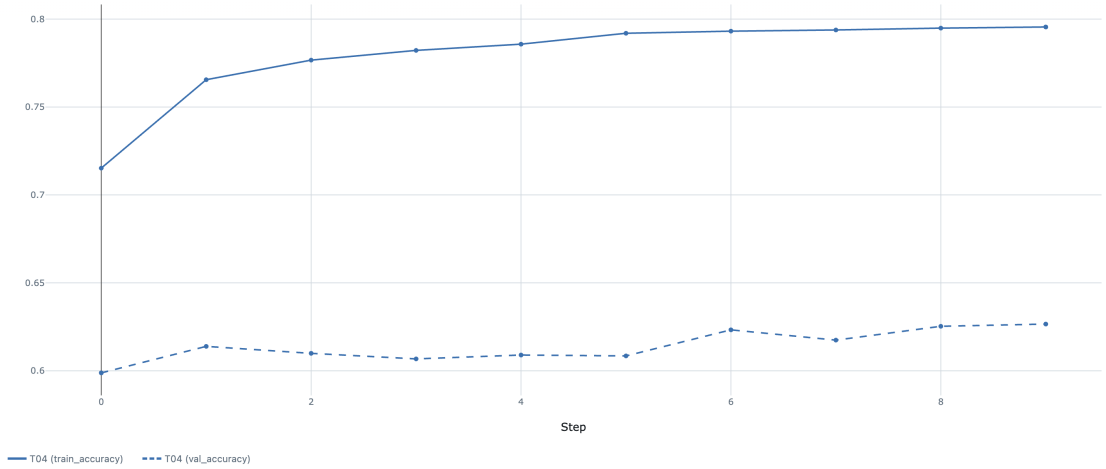


Figure 15: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T04.

In addition, the classification report from the test's final epoch has been illustrated in Table 6.

Table 6: Classification report from the final epoch. Test ID: T04.

Class	Precision	Recall	F1-Score	Support
Cloud	0.00	0.00	0.00	46921
Land	0.68	0.76	0.72	753582
Sea	0.53	0.49	0.51	505529
Accuracy			0.63	1306032
Macro Avg	0.40	0.42	0.41	1306032
Weighted Avg	0.60	0.63	0.61	1306032

Lastly, the last epoch's confusion matrix has been illustrated in Figure 16.

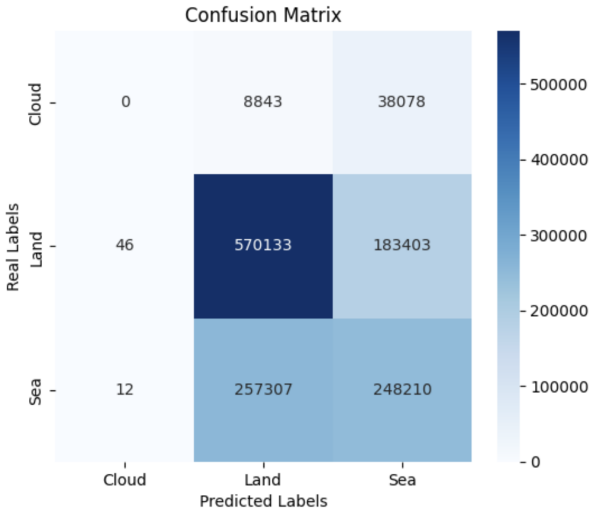


Figure 16: Confusion matrix from the final epoch. Test ID: T05.

3.1.5 T05

The train- and evaluation accuracy per epoch is plotted in Figure 17.

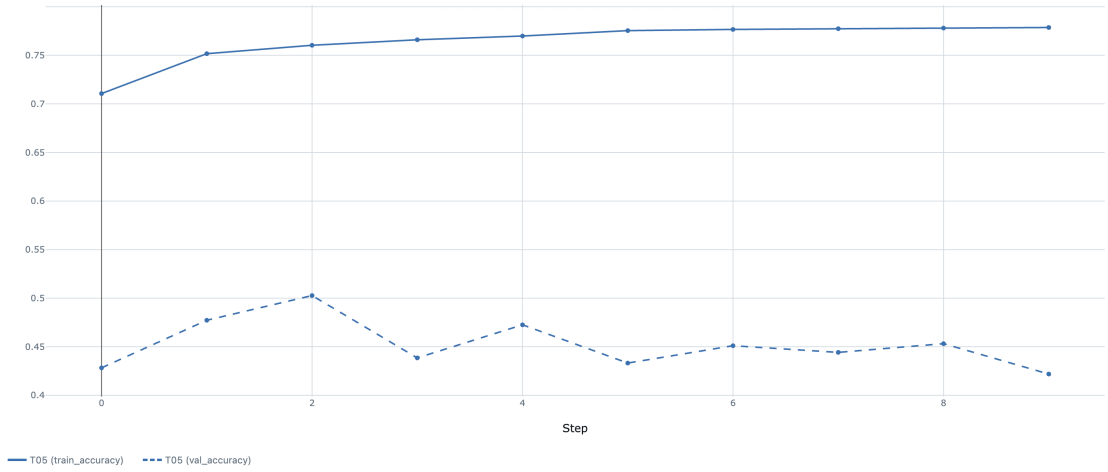


Figure 17: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T05.

In addition, the classification report from the test's final epoch has been illustrated in Table 7.

Table 7: Classification report from the final epoch. Test ID: T05.

Class	Precision	Recall	F1-Score	Support
Cloud	0.17	0.09	0.12	380366
Land	0.42	0.78	0.55	634497
Sea	0.51	0.31	0.39	944185
Accuracy			0.42	1959048
Macro Avg	0.37	0.39	0.35	1959048
Weighted Avg	0.41	0.42	0.39	1959048

Lastly, the last epoch's confusion matrix has been illustrated in Figure 18.

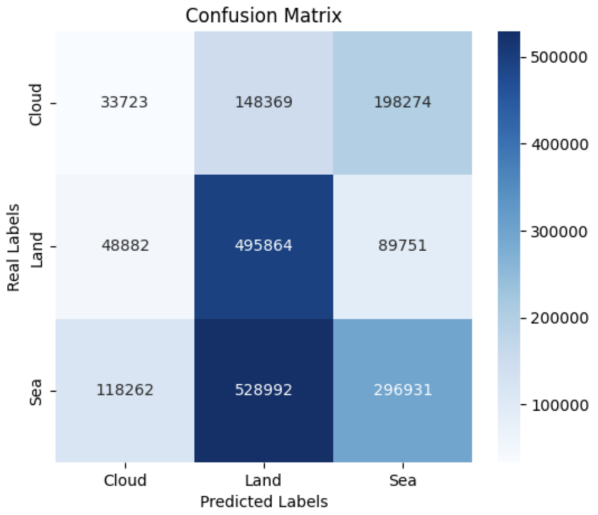


Figure 18: Confusion matrix from the final epoch. Test ID: T05.

3.1.6 T06

The train- and evaluation accuracy per epoch is plotted in Figure 19.

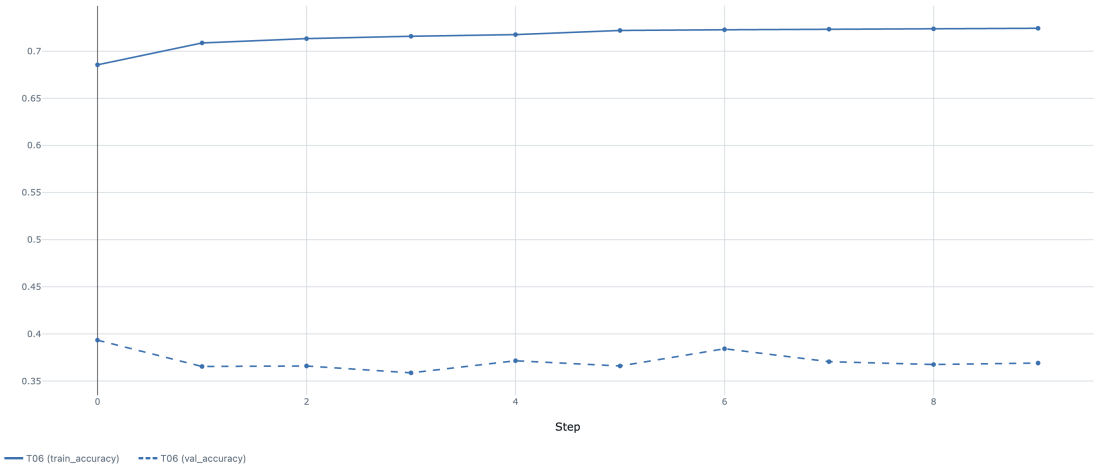


Figure 19: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T06.

In addition, the classification report from the test's final epoch has been illustrated in Table 8.

Table 8: Classification report from the final epoch. Test ID: T06.

Class	Precision	Recall	F1-Score	Support
Cloud	0.44	0.34	0.39	587647
Land	0.23	0.29	0.26	526452
Sea	0.44	0.44	0.44	844949
Accuracy			0.37	1959048
Macro Avg	0.37	0.36	0.36	1959048
Weighted Avg	0.38	0.37	0.37	1959048

Lastly, the last epoch's confusion matrix has been illustrated in Figure 20.

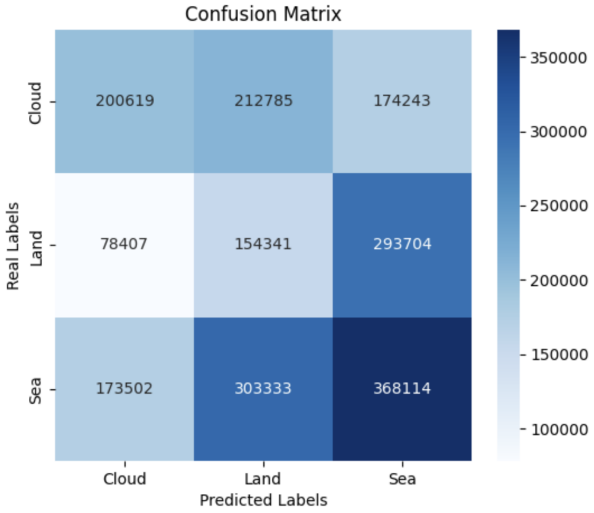


Figure 20: Confusion matrix from the final epoch. Test ID: T05.

3.2 Sources of Error

Now that all tests have been presented, the sources of error can be introduced. One major source is the corruption of the ENVI images. ENVI cannot achieve 100% accuracy when labeling images, as the process is semi-automatic. After plotting the labeled images, several errors were observed. The most notable problem is that some sea pixels, especially near clouds over the sea, have been incorrectly labeled as land when they are actually sea or cloud. An example image illustrating this issue is shown in Figure 21[2].

[2]All images are shown in appendix D.

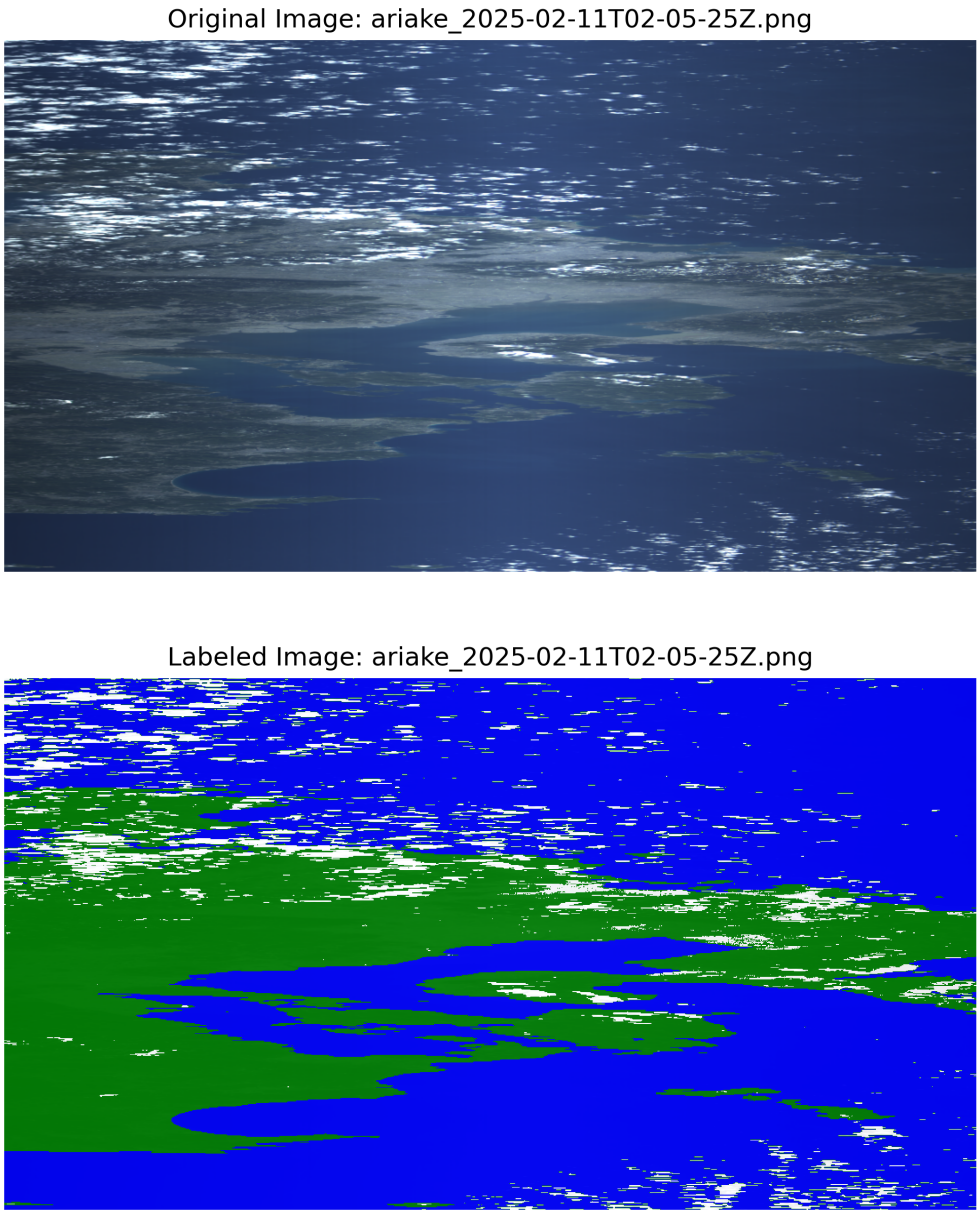


Figure 21: Original image and its plotted labeled image created with ENVI. Blue = sea, green = land, white = cloud. Note the several sea/cloud pixels that have been labeled as land.

Additionally, the original code of 1D-JustoLiuNet is structured in a way that may lead to misunderstandings or overlooked details during development, as it relies heavily on comments rather than a clearly organized codebase [9].

3.3 Why the 1D-JustoLiuNet Doesn't Work as Intended

One important factor contributing to the model's lower-than-expected performance is the use of min-max normalization applied individually to each image, band by band. While this approach ensures that each image is scaled appropriately within its own dynamic range, it can unintentionally reduce the natural variability between images. For example, a sea pixel from one location and a sea pixel from another location may

originally have significantly different spectral signatures due to differences in lighting, vegetation, or environmental conditions (see Appendix D for example images). However, because normalization is performed separately for each image, the relative differences between images can be compressed, making it harder for the model to distinguish meaningful global patterns across the entire dataset. Although per-image normalization is a standard practice in hyperspectral data processing, in this case it may have contributed to the model's difficulty in generalizing across diverse images.

Another important factor is the presence of corrupted images in the training data, as discussed in Section 3.2. These corrupted images may have introduced additional noise into the learning process, confusing the model and forcing it to learn from partially degraded or misleading data. However, although data corruption is certainly a contributing factor, it alone cannot account for the significant gap between the performance observed in this project and the 93% evaluation accuracy reported in the article *Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN* [5, page 4]. It is important to emphasize that the dataset used in that article consisted primarily of relatively "easy" images: scenes without snow, minimal environmental variation, and overall high visual similarity across the dataset [10]. In contrast, the datasets used in this project featured much greater diversity, both geographically and environmentally, placing a far heavier burden on the model's ability to generalize. This fundamental difference in dataset complexity likely explains a large part of the discrepancy in validation accuracy.

A third notable observation throughout all tests is the model's strong bias toward the class with the highest number of samples. Across every experiment, the class with the largest support consistently achieved the highest F1-score, while the class with the fewest examples suffered from very low precision and recall. This imbalance suggests that the model prioritizes majority classes during learning and struggles to correctly classify underrepresented classes, especially when the total amount of training data is relatively small.

Another factor that likely contributed to the limited generalization ability is the model's use of standard ReLU activation functions without incorporating dropout layers. Standard ReLU, while effective in many settings, is prone to neuron "death," where neurons stop activating entirely if they consistently receive negative inputs during training. This can reduce the network's capacity to learn a wide range of features, particularly when the dataset is small or highly varied. Using a Leaky ReLU activation function could have mitigated this issue by allowing small, nonzero gradients even for negative inputs, keeping more neurons active and contributing to the learning process [2, page 193]. Moreover, the complete absence of dropout layers meant that the model was free to memorize specific patterns from the training set without being encouraged to develop more broadly applicable feature representations. Dropout acts as a regularizer that forces the network to spread information across different neurons, improving its robustness to new and unseen inputs [2, page 258-261]. Without these mechanisms, the model was highly prone to overfitting the particular characteristics of the training data. In practice, this led the network to specialize in recognizing patterns from the training images rather than developing the generalization abilities needed to perform well on more diverse and challenging evaluation data.

In summary, the model's limited performance can be attributed to a combination of factors: the compression of natural spectral differences due to per-image min-max nor-

malization, the presence of corrupted and inconsistent training labels, the increased complexity and variability of the evaluation datasets compared to previous studies, and architectural choices that led to specialization rather than generalization. Together, these issues prevented the model from achieving strong validation results on more diverse and challenging data.

3.4 Future Work and Recommendations

One of the primary observations from this project is that relying solely on spectral signatures for classification becomes increasingly difficult when the images exhibit large variations in environmental conditions, such as lighting, atmospheric effects, and geographic features. While spectral information alone can perform well under consistent and controlled conditions, its effectiveness drops significantly when faced with highly diverse and complex datasets.

To address this, future work should consider incorporating spatial information alongside spectral features. Instead of treating each pixel independently, models could be designed to also consider the neighboring pixel context, allowing them to capture local textures and patterns that purely spectral information might miss. This could be achieved by moving from 1D CNNs operating on spectral data, to 2D or even 3D CNN architectures that process spatial and spectral data together.

Another valuable direction would be to directly test the model's performance on the dataset from the article *Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN*, not just compare validation accuracies reported. This would allow for a fairer evaluation of whether the model struggles primarily because of differences in dataset difficulty or due to architectural limitations.

4 Conclusion

This project set out to adapt and evaluate the 1D-JustoLiuNet for the classification of hyperspectral images from the HYPSO-2 satellite. Through extensive testing, it was found that while the model can learn meaningful patterns under controlled conditions, its performance drops significantly on more diverse and challenging datasets. Several factors contributed to this outcome, including per-image min-max normalization compressing spectral variability, corrupted and inconsistent training labels, increased dataset complexity, and architectural limitations favoring specialization over generalization.

The findings highlight the challenges of relying solely on spectral information for pixel-wise classification when working with real-world satellite data characterized by large environmental variability. Future improvements should focus on combining spectral and spatial features, exploring alternative CNN architectures, and performing fairer benchmarking against previous datasets.

Despite the limitations observed, the work presented here provides valuable insights into the specific difficulties of hyperspectral segmentation for operational satellite use and lays the foundation for future improvements in both methodology and model design.

References

[1] ePortals. *Hyperspectral Imaging*. Version 1.0. 2023.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[3] Izaak Neutelings. *Neural Networks*. 2025. URL: https://tikz.net/neural_networks/#full_code.

[4] Janosh Riebesell. *Convolution Operator*. 2025. URL: https://tikz.net/conv2d/.

[5] Jon Alvarez Justo et al. "Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN". In: *Remote Sensing* 17.4 (2025), p. 642. DOI: 10.3390/rs17040642. URL: https://doi.org/10.3390/rs17040642.

[6] Ilaria Cacciari and Anedio Ranfagni. *Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users*. Version 1.0. 2024.

[7] CoderzColumn. *PyTorch: Learning Rate Schedules*. 2022. URL: https://coderzcolumn.com/tutorials/artificial-intelligence/pytorch-learning-rate-schedules.

[8] Jorge Félix Martínez Pazos. *GPU Accelerated Deep Learning with PyTorch on Windows*. 2024. URL: https://medium.com/@jorgefmp.mle/gpu-accelerated-deep-learning-with-pytorch-on-windows-519898e4c283.

[9] Jon Alvarez Justo et al. "Sea-Land-Cloud Segmentation in Satellite Hyperspectral Imagery by Deep Learning". In: *arXiv preprint arXiv:2310.16210* (2023).

[10] Jon A Justo et al. "An Open Hyperspectral Dataset with Sea-Land-Cloud Ground-Truth from the Hypso-1 Satellite". In: *2023 13th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE. 2023, pp. 1–5.

A Code

The code is divided into multiple files, where each file complements the others, and the final execution is performed in `train.py`. If you want to run the code, clone this GitHub repository. In addition, the code is provided below.

For a better understanding of the code, please see Section 2.10.

A.1 `libraries.py`

```python
# libraries.py - Common file for all libraries

# System libraries
import os
import sys
import time
import random
import warnings

# Numerical calculations and data processing
import numpy as np
import pandas as pd
import cupy as cp
import csv
import random as rd
from pathlib import Path

# Visualization
import matplotlib.pyplot as plt
from matplotlib import image as mpimg
import seaborn as sns
from PIL import Image
import matplotlib.colors as mcolors
import matplotlib.patches as mpatches
import spectral

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
↪ QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score, classification_report,
↪ confusion_matrix
from sklearn.decomposition import PCA
import joblib

# Deep Learning (PyTorch, Keras)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from torch.cuda.amp import autocast, GradScaler
import torch.multiprocessing as mp
from torch.cuda import amp
from torch.optim.lr_scheduler import ReduceLROnPlateau

# Weight Adjusters
from collections import Counter

# Optimizers
from ranger_adabelief import RangerAdaBelief

# HYPSO Package
import hypso
from hypso import Hypso2
from hypso.load import load_l1a_nc_cube # Raw
from hypso.load import load_l1b_nc_cube # Radiance
from hypso.load import load_l1c_nc_cube # Reflectance
from hypso.load import load_l1d_nc_cube # Reflectance

# Progress Bar
from tqdm import tqdm

# Cache
sys.dont_write_bytecode = True
torch.cuda.empty_cache()

# Logger
import logging
logging.basicConfig(level=logging.DEBUG)
logging.getLogger('matplotlib').setLevel(logging.ERROR)
from termcolor import colored
warnings.filterwarnings("ignore")
import mlflow
import mlflow.pytorch

print("Libraries are loaded!")
```

A.2 cnn_1d.py

```
import sys
import os

```

```
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from libraries import *

#########################################################
#########################################################
#########################################################

class JustoLiuNet1D_torch(nn.Module):
    def __init__(self, num_features, num_classes=3, kernel_size=6, starting_kernels=6):
        super(JustoLiuNet1D_torch, self).__init__()

        self.conv1 = nn.Conv1d(1, starting_kernels, kernel_size=kernel_size)
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.conv2 = nn.Conv1d(starting_kernels, starting_kernels * 2, kernel_size=kernel_size)
        self.conv3 = nn.Conv1d(starting_kernels * 2, starting_kernels * 3, kernel_size=kernel_size)
        self.conv4 = nn.Conv1d(starting_kernels * 3, starting_kernels * 4, kernel_size=kernel_size)

        self.classifier = nn.Linear(48, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)

        x = F.relu(self.conv2(x))
        x = self.pool(x)

        x = F.relu(self.conv3(x))
        x = self.pool(x)

        x = F.relu(self.conv4(x))
        x = self.pool(x)

        #print("DEBUG - Shape before flatten:", x.shape)

        x = x.view(x.size(0), -1)

        if self.classifier is None:
            raise RuntimeError(f"'classifier' is not defined yet. Set Linear input dim to {x.size(1)} in __init__.")

        return self.classifier(x)
```

A.3 processing.py

```python
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from libraries import *

#########################################################
#########################################################
#########################################################

def load_image(image_path, HEIGHT=598, WIDTH=1092, BANDS=120):
    with open(image_path, 'rb') as f:
        image = cp.fromfile(f, dtype=cp.uint16)
        image = cp.asnumpy(image)
        image = image.reshape((BANDS, HEIGHT, WIDTH))
        image = image.transpose((1, 2, 0))
        image = image.reshape((-1, BANDS))

    image = cut_bands(image)

    return image

#########################################################

def load_label(label_path, HEIGHT=598, WIDTH=1092):
    label = np.fromfile(label_path, dtype=np.uint8)
    label = label.reshape((HEIGHT, WIDTH))
    label = label - 1
    label = label.flatten()
    return label

#########################################################

def global_min_max_normalization(images, verbose=False):
    """
    Normalize the images using global per-band min-max normalization. That is,
    the minimum and maximum values are calculated separately for each spectral band
    across all pixels in the dataset.
    """
    min_values = images.min(dim=0).values
    max_values = images.max(dim=0).values

    if verbose:
```

```
45          print(colored(f"Checking spectra of the first 10 pixels.",
            ↪  "red"))
46          check_normalization(images, "red")
47          print("\n")
48
49      new_images = (images - min_values) / (max_values - min_values +
        ↪  1e-8)
50
51      verify_global_min_max_normalization(images, new_images)
52
53      if verbose:
54          print("\n")
55          print(colored(f"Checking spectra of the first 10 pixels after
            ↪  min-max normalization.", "green"))
56          check_normalization(new_images, "green")
57
58      return new_images
59
60  ########################################################
61
62  def verify_global_min_max_normalization(original_images,
    ↪  normalized_images, epsilon=1e-8):
63      min_vals = original_images.min(dim=0).values
64      max_vals = original_images.max(dim=0).values
65
66      expected = (original_images - min_vals) / (max_vals - min_vals +
        ↪  epsilon)
67
68      if not torch.allclose(normalized_images, expected, atol=1e-6):
69          diffs = torch.abs(normalized_images - expected)
70          max_diff = diffs.max().item()
71          print(colored(f"Normalization mismatch! Maximum deviation:
            ↪  {max_diff}", "red"))
72          raise AssertionError("Normalization doesn't follow the
            ↪  formula.")
73      else:
74          print(colored("All pixels are correctly normalized.","green"))
75
76  ########################################################
77
78  def check_normalization(images, color):
79      br1 = 0
80      for i in range(images.shape[0]):
81          print(colored(images[i], color))
82          if br1 == 10:
83              break
84          br1 += 1
85
86  ########################################################
87
```

```python
def cut_bands(image, trim_start=3, trim_end=117):
    return image[:, trim_start:trim_end]

#########################################################

class normalization_manager:
    def __init__(self, epsilon=1e-8):
        self.min_vals = None
        self.max_vals = None
        self.epsilon = epsilon

    def fit(self, images):
        self.min_vals = images.min(dim=0).values
        self.max_vals = images.max(dim=0).values
        print(colored("Fitted normalization parameters from training data.", "green"))

    def transform(self, images, verify=False, verbose=False):
        if self.min_vals is None or self.max_vals is None:
            raise ValueError("NormalizationManager not fitted. Call fit(images) first.")

        if verbose:
            print(colored(f"Checking spectra of the first 10 pixels.", "red"))
            self.check_normalization(images, "red")
            print("\n")

        norm_images = (images - self.min_vals) / (self.max_vals - self.min_vals + self.epsilon)

        if verify:
            self.verify(images, norm_images)

        if verbose:
            print("\n")
            print(colored(f"Checking spectra of the first 10 pixels after min-max normalization.", "green"))
            self.check_normalization(norm_images, "green")

        return norm_images

    def verify(self, original_images, normalized_images):
        expected = (original_images - self.min_vals) / (self.max_vals - self.min_vals + self.epsilon)

        if not torch.allclose(normalized_images, expected, atol=1e-6):
            diffs = torch.abs(normalized_images - expected)
            max_diff = diffs.max().item()
```

```
            print(colored(f"Normalization mismatch! Maximum deviation:
            ↪ {max_diff}", "red"))
            raise AssertionError("Normalization doesn't follow the
            ↪ formula.")
        else:
            print(colored("All pixels are correctly
            ↪ normalized.","green"))

    def check_normalization(self, images, color):
        br1 = 0
        for i in range(images.shape[0]):
            print(colored(images[i], color))
            if br1 == 10:
                break
            br1 += 1
```

A.4 train_functions.py

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪ '..')))

from libraries import *
from functions.processing import load_label

#########################################################
#########################################################
#########################################################

def train_loop(model, train_loader, val_loader, criterion, optimizer,
↪ scheduler, device,
               save_path="models/best_model.pth", num_epochs=30):
    best_accuracy = 0.0

    classes = ["Cloud", "Land", "Sea"]
    print(f"DEBUG - Number of training batches: {len(train_loader)}")

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0.0
        labels_per_epoch = []
        predictions_per_epoch = []

        loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}",
        ↪ leave=False, colour="red")

```

```
        # Training
        train_accuracy, total_loss = train_subloop(
            loop, model, criterion, optimizer, device,
            predictions_per_epoch, labels_per_epoch, total_loss
            )

        # Evaluation
        model.eval()
        all_labels_eval = []
        all_preds_eval = []

        with torch.no_grad():
            all_labels_eval, all_preds_eval, val_loss = eval_subloop(
                val_loader, model, criterion, device,
                all_labels_eval, all_preds_eval
                )

        val_accuracy = accuracy_score(all_labels_eval, all_preds_eval)

        # Print results
        print("Unique labels in eval-data:",
        ↪  sorted(set(all_labels_eval)))
        print("Unique predictions:", sorted(set(all_preds_eval)))
        print(colored(f"TRAIN - Epoch {epoch+1}, Loss:
        ↪  {total_loss:.4f}, Accuracy: {train_accuracy:.2f}%",
        ↪  "magenta"))
        print(colored(f"EVAL  - Epoch {epoch+1}, Accuracy:
        ↪  {val_accuracy*100:.2f}%", "green"))
        print(classification_report(all_labels_eval, all_preds_eval,
        ↪  target_names=classes))

        # Log MLflow
        log_mlflow_train(train_accuracy, val_accuracy, total_loss,
        ↪  val_loss, epoch)
        log_classification_report_mlflow(classification_report(
        all_labels_eval, all_preds_eval, target_names=classes,
        ↪  output_dict=True),epoch)

        # Save model
        if val_accuracy > best_accuracy:
            best_accuracy = val_accuracy
            save_model(model, best_accuracy, save_path)
            print(f"Model saved with accuracy: {best_accuracy:.2f}%")

        # Confusion Matrix
        confusion_matrix_plot(all_labels_eval, all_preds_eval, classes,
        ↪  epoch)

        scheduler.step()
```

```
        print(colored(f"Learning rate:
        ↪  {scheduler.get_last_lr()[0]:.6f}", "yellow"))

        print("\n")

#########################################################

def train_subloop(loop, model, criterion, optimizer, device,
                  predictions_per_epoch, labels_per_epoch, total_loss):
    correct = 0
    total = 0
    for batch in loop:
        spectrum, labels = batch
        spectrum = spectrum.unsqueeze(1).to(device, non_blocking=True)
        labels = labels.view(-1).to(device, non_blocking=True)

        optimizer.zero_grad()
        output = model(spectrum)

        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

        predictions_per_epoch.append(output.detach().cpu())
        labels_per_epoch.append(labels.detach().cpu())

        total_loss += loss.item()
        _, predicted = output.max(1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_accuracy = 100 * correct / total

    return train_accuracy, total_loss

#########################################################

def eval_subloop(val_loader, model, criterion, device,
                 all_labels_eval, all_preds_eval):
    val_loss = 0.0
    for spectrum, labels in tqdm(val_loader, desc="Evaluation",
    ↪  leave=True, colour="blue"):
        spectrum = spectrum.unsqueeze(1).to(device)
        labels = labels.view(-1).to(device)
        output = model(spectrum)
        _, predicted = torch.max(output, 1)
        loss = criterion(output, labels)

        all_labels_eval.extend(labels.cpu().numpy())
        all_preds_eval.extend(predicted.cpu().numpy())
```

```
117         val_loss += loss.item()
118 
119     return all_labels_eval, all_preds_eval, val_loss
120 
121 ########################################################
122 
123 def confusion_matrix_plot(all_labels_eval, all_preds_eval, classes,
    ↪  epoch):
124     cm = confusion_matrix(all_labels_eval, all_preds_eval)
125     plt.figure(figsize=(6, 5))
126     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
        ↪  xticklabels=classes, yticklabels=classes)
127     plt.xlabel("Predicted Labels")
128     plt.ylabel("Real Labels")
129     plt.title("Confusion Matrix")
130 
131     output_dir = "plots/validation_plots"
132     os.makedirs(output_dir, exist_ok=True)
133     filename = f"confusion_matrix_EPOCH_{epoch+1}.png"
134     filepath = os.path.join(output_dir, filename)
135     plt.savefig(filepath)
136     plt.close()
137 
138     mlflow.log_artifact(filepath, artifact_path="confusion_matrices")
139 
140 ########################################################
141 
142 def save_model(model, best_accuracy, save_path):
143     torch.save({
144         'model_state_dict': model.state_dict(),
145         'best_accuracy': best_accuracy
146     }, save_path)
147 
148 ########################################################
149 
150 class FocalLoss(nn.Module):
151     def __init__(self, alpha=None, gamma=2, reduction='mean'):
152         super(FocalLoss, self).__init__()
153         self.alpha = alpha  # Forventer tensor med shape [n_classes]
154         self.gamma = gamma
155         self.reduction = reduction
156 
157     def forward(self, inputs, targets):
158         ce_loss = F.cross_entropy(inputs, targets, reduction='none')
159         p_t = torch.exp(-ce_loss)
160 
161         if self.alpha is not None:
162             # Hent riktig alpha for hver klasse i batchen
163             alpha = self.alpha.gather(0, targets)  # Shape:
                ↪  [batch_size]
```

```
            focal_loss = alpha * (1 - p_t) ** self.gamma * ce_loss
        else:
            focal_loss = (1 - p_t) ** self.gamma * ce_loss

        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        return focal_loss

########################################################

def get_class_weights(label_paths, num_classes):
    all_labels = []

    for path in label_paths:
        labels = load_label(path)
        all_labels.extend(labels)

    labels_tensor = torch.tensor(all_labels, dtype=torch.long)
    class_counts = torch.bincount(labels_tensor,
    ↪  minlength=num_classes).float()

    # No zero division
    epsilon = 1e-6
    class_counts += epsilon

    print("Using class weights.")

    # Calculate weights
    alpha = (1.0 / class_counts)
    alpha = alpha / alpha.sum() # Normalization

    print(colored(f"Successfully calculated class weights: {alpha}",
    ↪  "green"))

    return alpha

########################################################

def log_mlflow_pre_train(EPOCHS, BATCH_SIZE, LR, LABEL_SMOOTHING,
                KERNEL_SIZE, STARTING_KERNELS, NUM_FEATURES,
                ↪  NUM_CLASSES,
                optimizer, scheduler, criterion, model, train_dataset,
                ↪  eval_dataset):
    mlflow.log_param("EPOCHS", EPOCHS)
    mlflow.log_param("BATCH_SIZE", BATCH_SIZE)
    mlflow.log_param("LR", LR)
    mlflow.log_param("LABEL_SMOOTHING", LABEL_SMOOTHING)
    mlflow.log_param("KERNEL_SIZE", KERNEL_SIZE)
```

```
    mlflow.log_param("STARTING_KERNELS", STARTING_KERNELS)
    mlflow.log_param("NUM_FEATURES", NUM_FEATURES)
    mlflow.log_param("NUM_CLASSES", NUM_CLASSES)
    mlflow.log_param("optimizer", optimizer.__class__.__name__)
    mlflow.log_param("scheduler", scheduler.__class__.__name__)
    mlflow.log_param("loss_function", criterion.__class__.__name__)
    mlflow.log_param("model", model.__class__.__name__)
    mlflow.log_param("train_dataset_size", len(train_dataset)/653016)
    mlflow.log_param("eval_dataset_size", len(eval_dataset)/653016)

########################################################

def log_mlflow_train(train_accuracy, val_accuracy, train_loss,
↪  val_loss, epoch):
    mlflow.log_metric(f"train_accuracy", train_accuracy/100,
    ↪  step=epoch)
    mlflow.log_metric(f"val_accuracy", val_accuracy, step=epoch)
    mlflow.log_metric(f"train_loss", train_loss, step=epoch)
    mlflow.log_metric(f"val_loss", val_loss, step=epoch)

########################################################

def log_classification_report_mlflow(report_dict, epoch):
    for class_name, metrics in report_dict.items():
        if isinstance(metrics, dict):
            for metric_name, value in metrics.items():
                if "support" or "accuracy" not in metric_name:
                    mlflow.log_metric(f"{class_name}_{metric_name}",
                    ↪  value, step=epoch)
        else:
            if "support" or "accuracy" not in class_name:
                mlflow.log_metric(f"{class_name}", metrics, step=epoch)
```

A.5 dataset.py

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪  '..')))

from libraries import *
from functions.processing import *
from manage_data import read_csv_file

########################################################
########################################################
########################################################
```

```
13
14  class merged_hyperspectral_dataset(Dataset):
15      def __init__(self, list_of_images, list_of_labels=None,
        ↪  normalizer=None):
16          all_images = []
17
18          if list_of_labels is not None:
19              all_labels = []
20
21          for image_path, label_path in tqdm(zip(list_of_images,
            ↪  list_of_labels),
22                                             desc="Loading images and
                                               ↪  labels",
23                                             total=len(list_of_images)):
24              image = load_image(image_path) # (pixels, bands)
25              all_images.append(image)
26
27              if list_of_labels is not None:
28                  label = load_label(label_path) # (pixels, )
29                  all_labels.append(label)
30
31          self.images = torch.from_numpy(
32          np.concatenate(all_images)).float().contiguous()
33
34          if list_of_labels is not None:
35              self.labels =
                ↪  torch.from_numpy(np.concatenate(all_labels)).long()
36          else:
37              self.labels = None
38
39          if normalizer is not None:
40              self.images = normalizer.transform(self.images,
                ↪  verify=True, verbose=False)
41
42      def __len__(self):
43          return self.images.shape[0]
44
45      def __getitem__(self, idx):
46          #print(self.images[idx].shape)
47          if self.labels is None:
48              return self.images[idx]
49          else:
50              return self.images[idx], self.labels[idx]
51
52  ##########################################################
53
54  """
55  bip_paths, dat_paths, png_paths = read_csv_file("csv/train_files.csv")
56
57  normalizer = normalization_manager()
```

```
raw_dataset = merged_hyperspectral_dataset(bip_paths, dat_paths)
normalizer.fit(raw_dataset.images)
dataset = merged_hyperspectral_dataset(
    bip_paths, dat_paths, normalizer=normalizer
)

dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

# Test en enkelt batch
for pixels, labels in dataloader:
    print(colored(f"Pixels shape: {pixels.shape}", "light_blue"))   # Forventet: (128, num_bands)
    print(colored(f"Labels shape: {labels.shape}", "light_green"))  # Forventet: (128,)
    print(colored(f"Unique labels in batch: {torch.unique(labels)}", "light_yellow"))
    break"
"""
```

A.6 manage_data.py

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from libraries import *

############################################################
############################################################
############################################################

"""
The functions move_forward(), move_back() and list_contents() are used for retrieving
.dat, .bip (/.bip@) and .png files for use in CSV creation.
"""

current_path = Path(".").resolve()

def move_forward(folder_name):
    global current_path
    new_path = current_path / folder_name
    if new_path.exists() and new_path.is_dir():
        current_path = new_path
        return f"Moved to: {current_path}"
```

```
25      return "Folder does not exist."
26
27  def move_back():
28      global current_path
29      if current_path.parent.exists():
30          current_path = current_path.parent
31          return f"Moved back to: {current_path}"
32      return "Cannot move back further."
33
34  def list_contents():
35      return [item.name for item in current_path.iterdir()]
36
37  ##########################################################
38
39  def get_dat_png_bip():
40      """
41      Collects and returns paths of .dat, .png, and .bip files from
        ↪ specified directories.
42      This function navigates through a directory structure to find and
        ↪ collect paths of files
43      with specific extensions (.dat, .png, .bip) from labeled and raw
        ↪ data directories.
44      Returns:
45          tuple: A tuple containing three lists:
46              - dat_paths (list): List of paths to .dat files.
47              - png_paths (list): List of paths to .png files.
48              - bip_paths (list): List of paths to .bip files.
49      """
50
51      dat_paths = []
52      dat_paths_checkup = []
53
54      png_paths = []
55      bip_paths = []
56
57      move_forward("corrected_labeled_data_CLOUD")
58      contents_labeled_data = list_contents()
59      for j in contents_labeled_data:
60          dat_paths.append((str(current_path / j))[35:])
61          dat_paths_checkup.append(j[:j.find("-l1a")])
62          #print(j[:j.find("-l1a")])
63
64      move_back()
65
66      #print(os.getcwd())
67
68
69      dat_paths_checkup = ([str(path) for path in dat_paths_checkup])
70      dat_paths_checkup = sorted(dat_paths_checkup)
71      #print(len(dat_paths_checkup))
```

```
    #print(dat_paths_checkup)

    move_forward("raw_data")
    L1_contents = list_contents()
    #print(L1_contents)

    print(dat_paths_checkup)

    for folder in L1_contents:
        move_forward(folder)
        L2_contents = list_contents()
        print(colored(L2_contents, "blue"))
        for folder_gr_2 in L2_contents:
            if folder_gr_2 in dat_paths_checkup:
                move_forward(folder_gr_2)
                L3_contents = list_contents()
                print(colored(L3_contents, "light_blue"))
                for i in L3_contents:
                    if i.endswith("Z.png"):
                        png_paths.append((str(current_path / i))[35:])
                    elif i.endswith(".bip"):
                        bip_paths.append((str(current_path / i))[35:])
                    elif i.endswith(".bip@"):
                        bip_paths.append((str(current_path / i))[35:])

                move_back()
        move_back()
    move_back()

    print("-"*100)
    print(sorted(png_paths))

    print(len(dat_paths), len(png_paths), len(bip_paths))

    return dat_paths, png_paths, bip_paths

########################################################

def create_csv_file():
    """
    Creates CSV files that maps .dat files
    to their corresponding .bip (/.bip@) and .png files.

    - Calls get_dat_png_bip() to retrieve paths.
    - Writes the file paths into 'train_files.csv' and
    'evaluate_files.csv' with headers: ["dat_files", "bip_files",
    ↪  "png_files"].
    """

```

```
120     dat_files_paths, png_files_paths, bip_files_paths =
        ↪ get_dat_png_bip()
121
122     dat_files_paths = sorted(dat_files_paths)
123     png_files_paths = sorted(png_files_paths)
124     bip_files_paths = sorted(bip_files_paths)
125
126     if len(dat_files_paths) == 1:
127         with open('csv/train_files.csv', mode='w') as train_file:
128             writer = csv.writer(train_file)
129             writer.writerow(["dat_files", "bip_files", "png_files"])
130             writer.writerow([dat_files_paths[0], bip_files_paths[0],
                ↪ png_files_paths[0]])
131     elif len(dat_files_paths) > 1:
132         split_index = int(len(dat_files_paths)*0.9)
133         train_dat_files = dat_files_paths[:split_index]
134         eval_dat_files = dat_files_paths[split_index:]
135
136         train_bip_files = bip_files_paths[:split_index]
137         eval_bip_files = bip_files_paths[split_index:]
138
139         train_png_files = png_files_paths[:split_index]
140         eval_png_files = png_files_paths[split_index:]
141
142         with open('csv/train_files.csv', mode='w') as train_file:
143             writer = csv.writer(train_file)
144             writer.writerow(["dat_files", "bip_files", "png_files"])
145             for i in range(len(train_dat_files)):
146                 writer.writerow([train_dat_files[i],
                    ↪ train_bip_files[i], train_png_files[i]])
147
148         with open('csv/evaluate_files.csv', mode='w') as eval_file:
149             writer = csv.writer(eval_file)
150             writer.writerow(["dat_files", "bip_files", "png_files"])
151             for i in range(len(eval_dat_files)):
152                 writer.writerow([eval_dat_files[i], eval_bip_files[i],
                    ↪ eval_png_files[i]])
153     else:
154         with open('csv/train_files.csv', mode='w') as train_file:
155             writer = csv.writer(train_file)
156             writer.writerow(["dat_files", "bip_files", "png_files"])
157             for i in range(len(dat_files_paths)):
158                 writer.writerow([dat_files_paths[i],
                    ↪ bip_files_paths[i], eval_png_files[i]])
159
160 ########################################################
161
162 def read_csv_file(csv_file_with_path):
163     """
164     Reads a CSV file containing .dat and .bip@ file paths.
```

```
    Parameters:
    - csv_file_with_path (str): Path to the CSV file.

    Returns:
    - bip_files (list): List of .bip (/.bip@) file paths.
    - dat_files (list): List of .dat file paths.
    - png_files (list): List of .png file paths.
    """

    dat_files = []
    bip_files = []
    png_files = []

    with open(f'{csv_file_with_path}', mode='r') as file:
        csvFile = csv.reader(file)

        i = 0
        for lines in csvFile:
            if i != 0:
                dat_files.append(lines[0])
                bip_files.append(lines[1])
                png_files.append(lines[2])
            i += 1

    return bip_files, dat_files, png_files

#########################################################

#create_csv_file()
```

A.7 train.py

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪  '..')))

from libraries import *
from manage_data import read_csv_file
from dataset import merged_hyperspectral_dataset
from functions.processing import normalization_manager
from functions.train_functions import train_loop
from functions.train_functions import get_class_weights
from functions.train_functions import FocalLoss
from functions.train_functions import log_mlflow_pre_train
from models.cnn_1d import JustoLiuNet1D_torch
```

```
15
16  ##########################################################
17  ##########################################################
18  ##########################################################
19
20  # MLflow
21  mlflow.set_experiment("CNN_hyperspectral_v1")
22
23  # Device
24  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
25  print(f"Device: {device}")
26
27  # Hyperparameters
28  EPOCHS = 10
29  BATCH_SIZE = 128
30  LR = 0.001
31  LABEL_SMOOTHING = 0.1
32  KERNEL_SIZE = 6
33  STARTING_KERNELS = 6
34  NUM_FEATURES = 114
35  NUM_CLASSES = 3
36
37  with mlflow.start_run():
38
39      # Data
40      train_bip_paths, train_labels_paths, _ =
        ↪  read_csv_file("csv/train_files.csv")
41      eval_bip_paths, eval_labels_paths, _ =
        ↪  read_csv_file("csv/evaluate_files.csv")
42
43      # Normalizer
44      normalizer = normalization_manager()
45      raw_train = merged_hyperspectral_dataset(train_bip_paths,
        ↪  train_labels_paths)
46      normalizer.fit(raw_train.images)
47      train_dataset = merged_hyperspectral_dataset(train_bip_paths,
        ↪  train_labels_paths, normalizer)
48      eval_dataset = merged_hyperspectral_dataset(eval_bip_paths,
        ↪  eval_labels_paths, normalizer)
49
50      # Dataloader
51      train_loader = DataLoader(train_dataset,
52                                batch_size=BATCH_SIZE,
53                                shuffle=True,
54                                num_workers=8,
55                                pin_memory=True)
56      eval_loader = DataLoader(eval_dataset,
57                               batch_size=BATCH_SIZE,
58                               shuffle=False,
59                               num_workers=8,
```

```
                                pin_memory=True)

    # Model
    model = JustoLiuNet1D_torch(num_features=NUM_FEATURES,
    ↪  num_classes=NUM_CLASSES,
    kernel_size=KERNEL_SIZE,
    ↪  starting_kernels=STARTING_KERNELS).to(device)
    total_params = sum(p.numel() for p in model.parameters())
    print(colored(f"Total parameters in {model.__class__.__name__}:
    ↪  {total_params}", "magenta"))

    optimizer = torch.optim.AdamW(model.parameters(), lr=LR,
    ↪  weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5,
    ↪  gamma=0.5)
    alpha = get_class_weights(train_labels_paths, NUM_CLASSES)
    #criterion = nn.CrossEntropyLoss(weight=alpha.to(device),
    ↪  label_smoothing=LABEL_SMOOTHING).to(device)
    criterion =
    ↪  nn.CrossEntropyLoss(label_smoothing=LABEL_SMOOTHING).to(device)
    #criterion = FocalLoss(alpha=alpha.to(device), gamma=3,
    ↪  reduction='sum').to(device)

    # Logging
    log_mlflow_pre_train(EPOCHS, BATCH_SIZE, LR, LABEL_SMOOTHING,
                KERNEL_SIZE, STARTING_KERNELS, NUM_FEATURES,
                NUM_CLASSES,optimizer, scheduler, criterion,
                model, train_dataset, eval_dataset)

    # Training
    print("Starting training...")
    train_loop(model, train_loader, eval_loader, criterion,
                optimizer, scheduler, device, num_epochs=EPOCHS)
    print("Training finished.")
```

B Complete Class Diagram

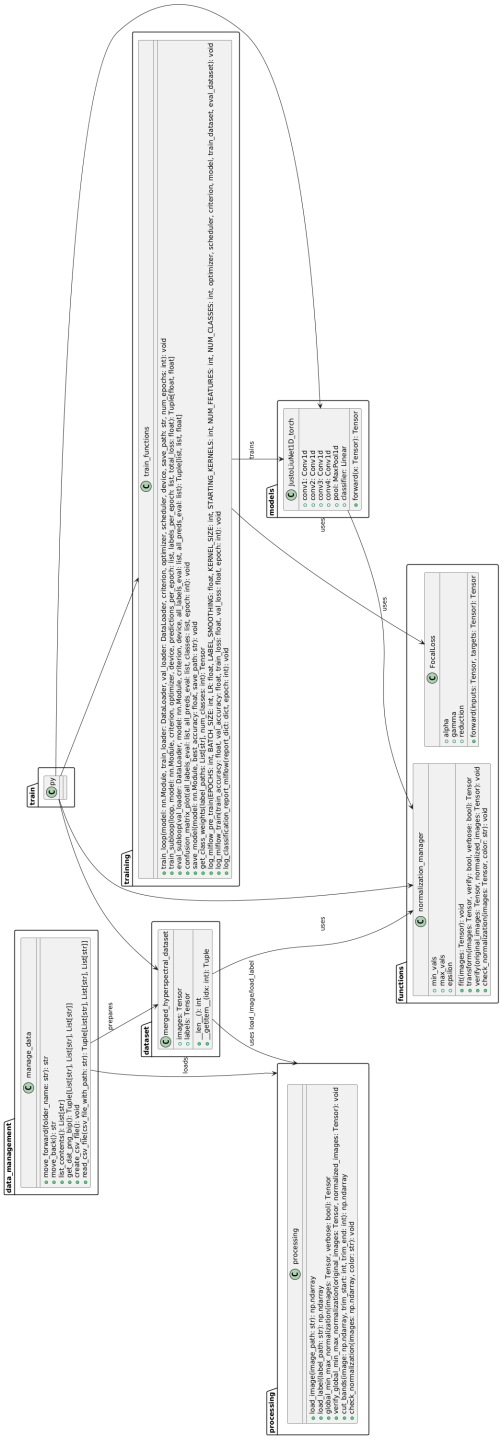


Figure 22: Complete class diagram of code.

C Images Used in Testing

Note here that the "Set" column in the tables below have either the variables: "T", "E" or "TE". "T" indicates that the image was in the training set. "E" indicates the image was in the evaluation set. Lastly, "TE" indicates the image was in both sets.

C.1 T01

Table 9: Images used in test T01.

Image Name	Date	Set
lacrau	2024-12-26	TE
bluenile	2025-01-25	TE
dardanelles	2025-03-08	TE
aquawatchlakehume	2025-03-08	TE
aeronetgalata	2025-01-02	TE
zeebrugge	2025-03-08	TE

C.2 T02

Table 10: Images used in test T02.

Image Name	Date	Set
lacrau	2024-12-26	TE
image61N6E	2025-03-13	TE
menindee	2025-01-02	TE
frohavet	2025-02-25	TE
gulfofcalifornia	2025-01-14	TE
gobabeb	2025-02-02	TE
aquawatchspencer	2025-01-03	TE
straitofgeorgia	2025-01-24	TE

C.3 T03

As stated in Table 2, there's 25 copies of this image during training, and the same image during evaluation.

Table 11: Images used in test T03.

Image Name	Date	Set
bluenile	2025-01-25	TE

C.4 T04

Table 12: Images used in test T04.

Image Name	Date	Set
lacrau	2024-12-26	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
aquawatchlakehume	2025-03-08	T
zeebrugge	2025-03-08	T
blanca	2025-02-04	T
aeronetgloria	2025-01-09	T
aeronetgalata	2025-01-02	E

C.5 T05

Table 13: Images used in test T05.

Image Name	Date	Set
lacrau	2024-12-26	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
menindee	2025-01-02	T
aeronetgalata	2025-01-02	T
aeronetgloria	2025-01-09	T
aquawatchgrippsland	2025-03-09	T
aquawatchlakehume	2025-03-08	T
image61N6E	2025-03-13	T
falklandsatlantic	2025-03-03	E
frohavet	2025-02-25	E
gobabeb	2025-02-02	E

C.6 T06

Table 14: Images used in test T06.

Image Name	Date	Set
aeronetgalata	2025-01-02	T
aeronetgloria	2025-01-09	T
aquawatchgrippsland	2025-03-09	T
aquawatchlakehume	2025-03-08	T
aquawatchmoreton	2025-01-22	T
aquawatchspencer	2025-01-03	T
ariake	2025-02-11	T
blanca	2025-02-04	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
erie	2025-03-13	T
falklandsatlantic	2025-03-03	T

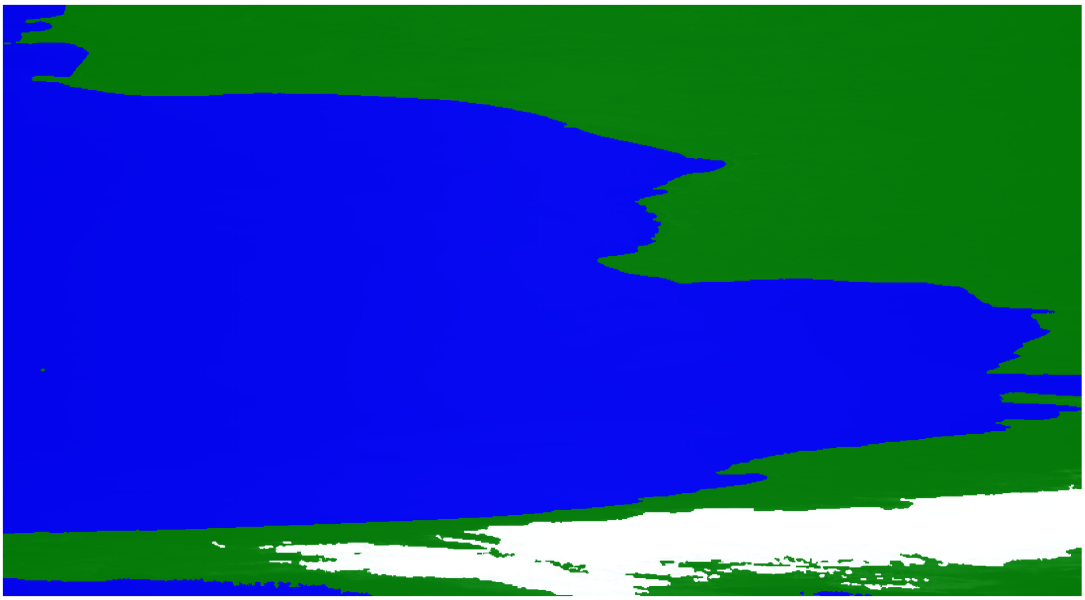
Image Name	Date	Set
frohavet	2025-02-25	T
gobabeb	2025-02-02	T
goddard	2025-01-09	T
grizzlybay	2025-01-22	T
gulfofcalifornia	2025-01-14	T
hypernetEstuary	2024-12-28	T
image61N6E	2025-03-13	T
image65N10E	2025-03-12	T
kemigawa	2025-01-22	T
lacrau	2024-12-26	T
longisland	2025-01-22	T
menindee	2025-01-02	T
moby	2025-01-08	E
straitofgeorgia	2025-01-24	E
zeebrugge	2025-03-08	E

D Used Images and Their Labels

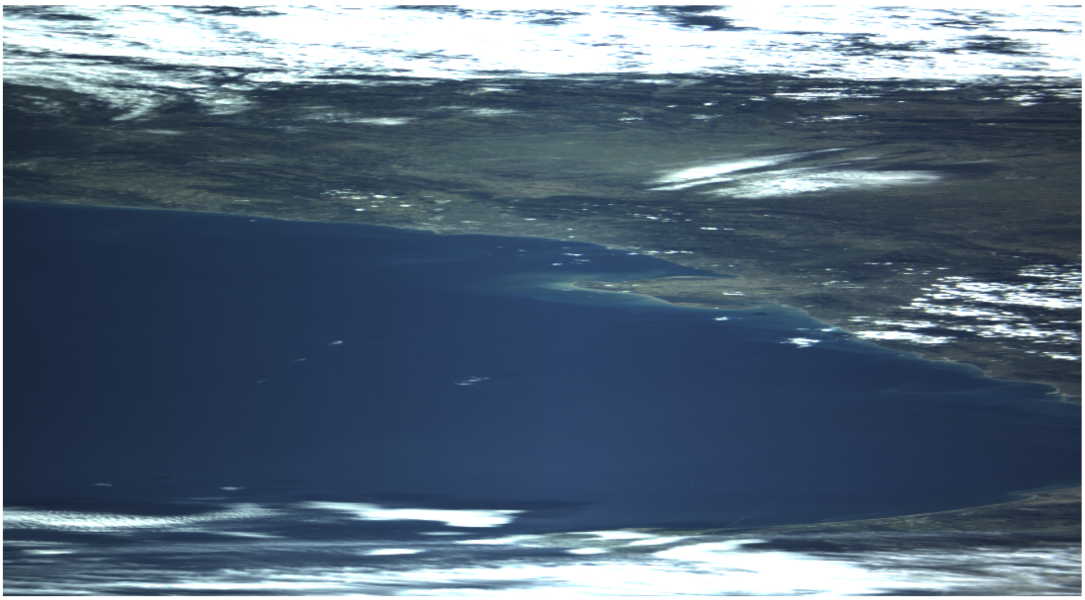
Note here that blue coloring is for sea, green is for land and white is for cloud.

Original Image: aeronetgalata_2025-01-02T08-52-34Z.png

Labeled Image: aeronetgalata_2025-01-02T08-52-34Z.png

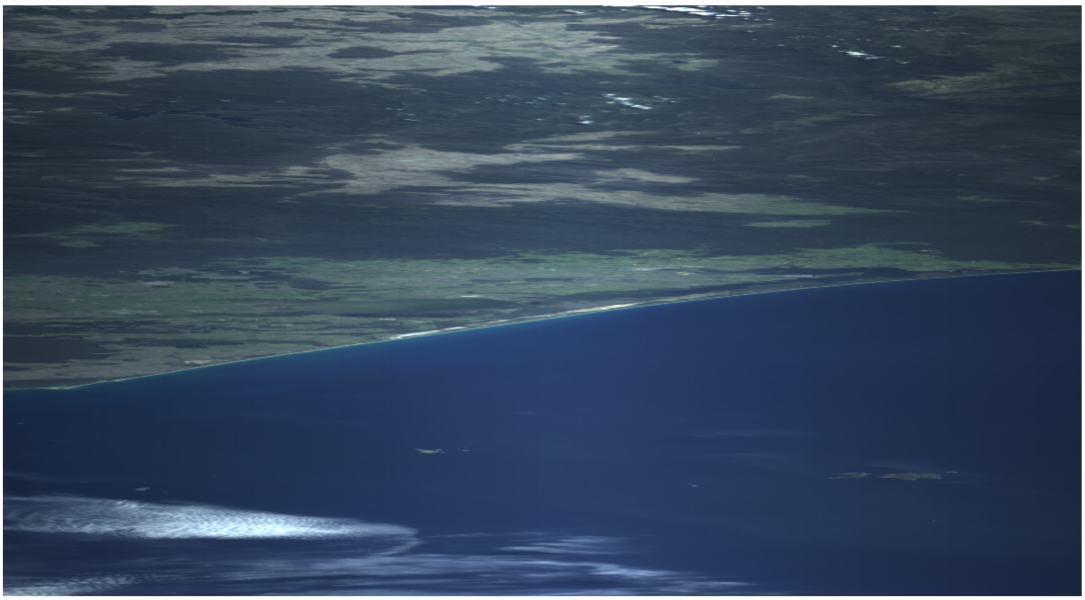


Original Image: aeronetgloria_2025-01-09T11-17-15Z.png



Labeled Image: aeronetgloria_2025-01-09T11-17-15Z.png

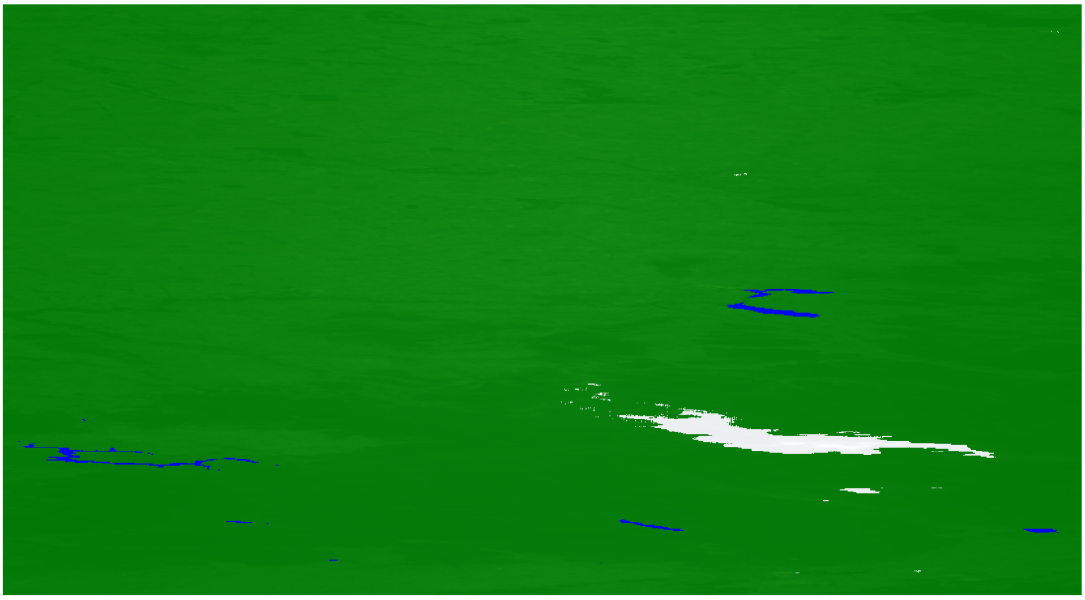
Original Image: aquawatchgrippsland_2025-03-09T00-16-17Z.png



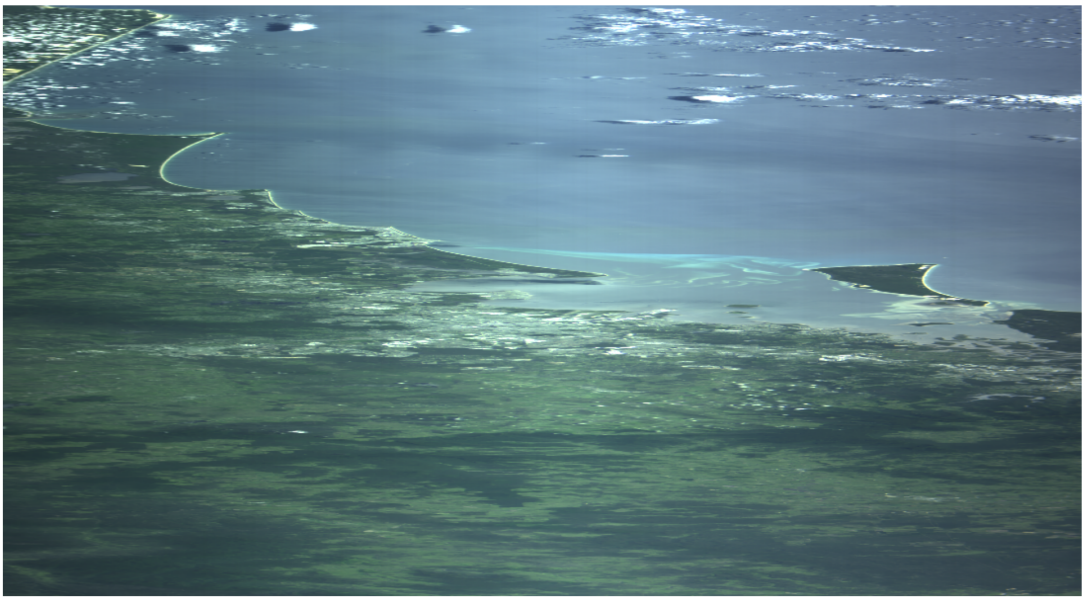
Labeled Image: aquawatchgrippsland_2025-03-09T00-16-17Z.png

Original Image: aquawatchlakehume_2025-03-08T00-09-52Z.png

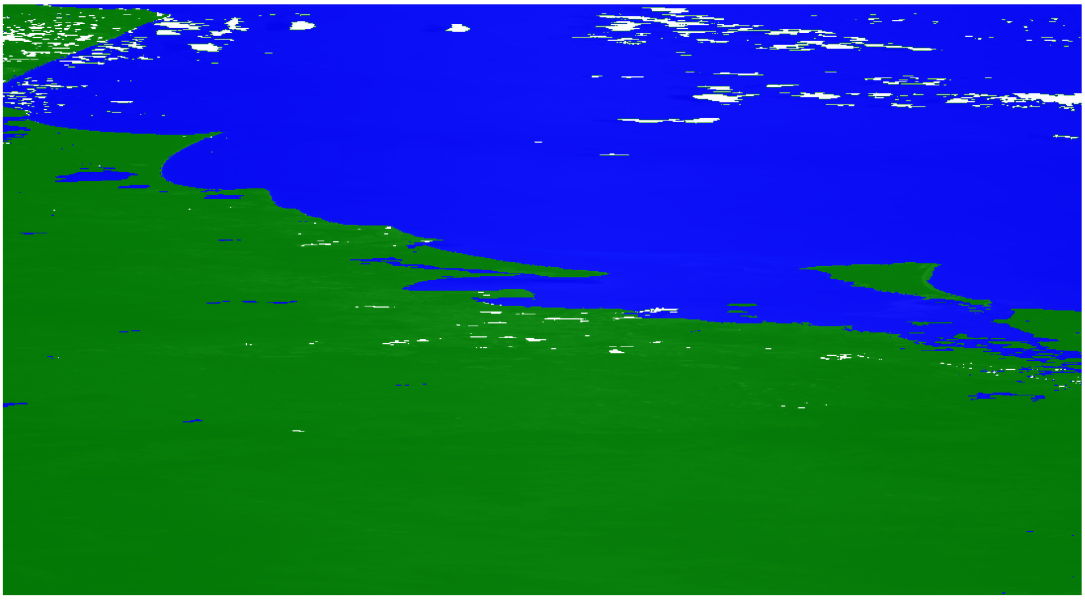
Labeled Image: aquawatchlakehume_2025-03-08T00-09-52Z.png



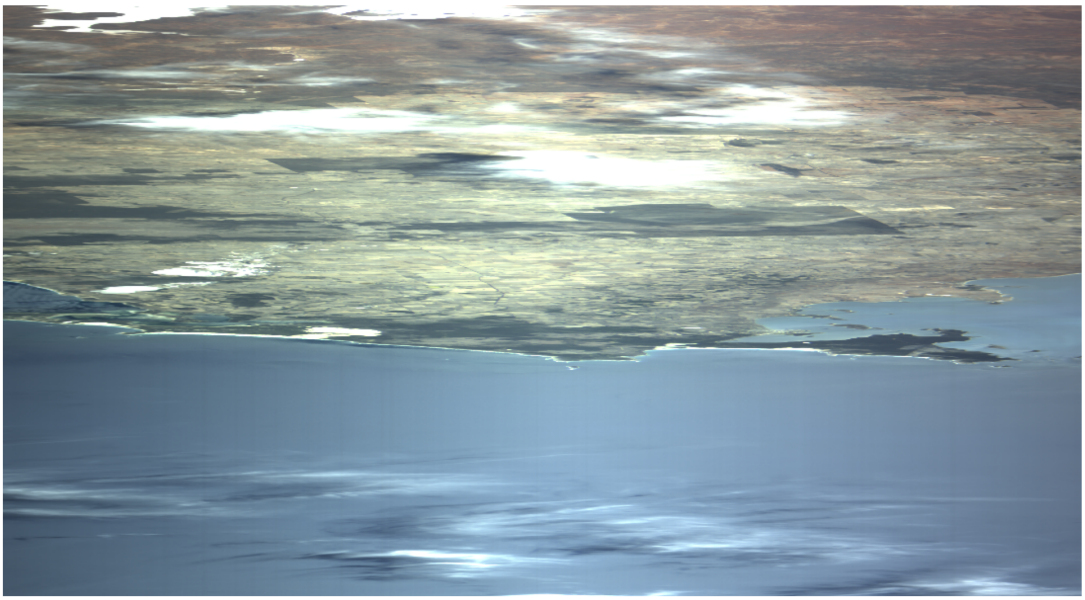
Original Image: aquawatchmoreton_2025-01-22T00-11-33Z.png



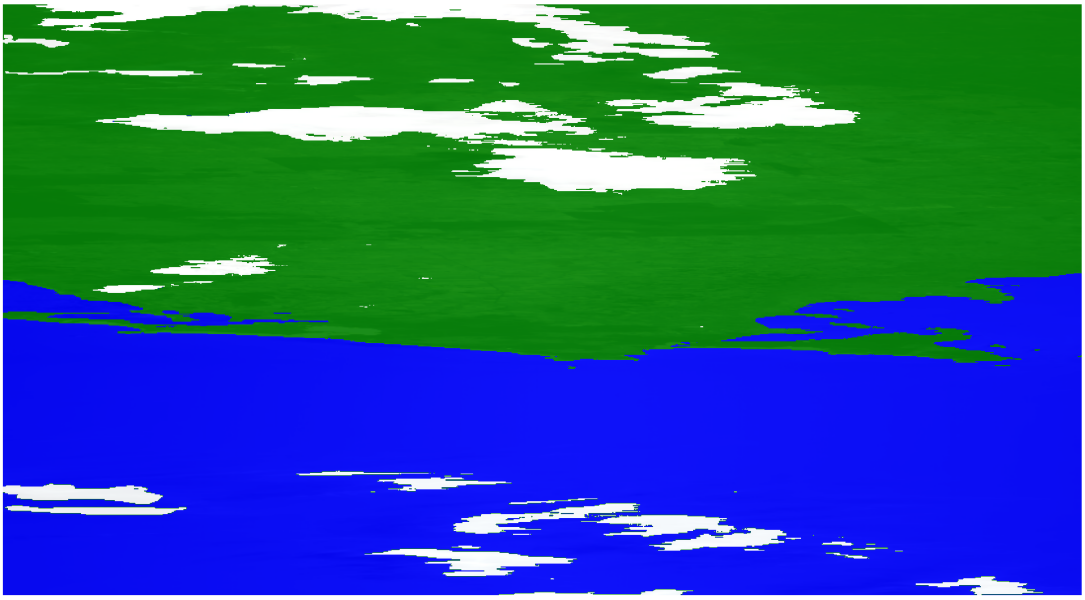
Labeled Image: aquawatchmoreton_2025-01-22T00-11-33Z.png



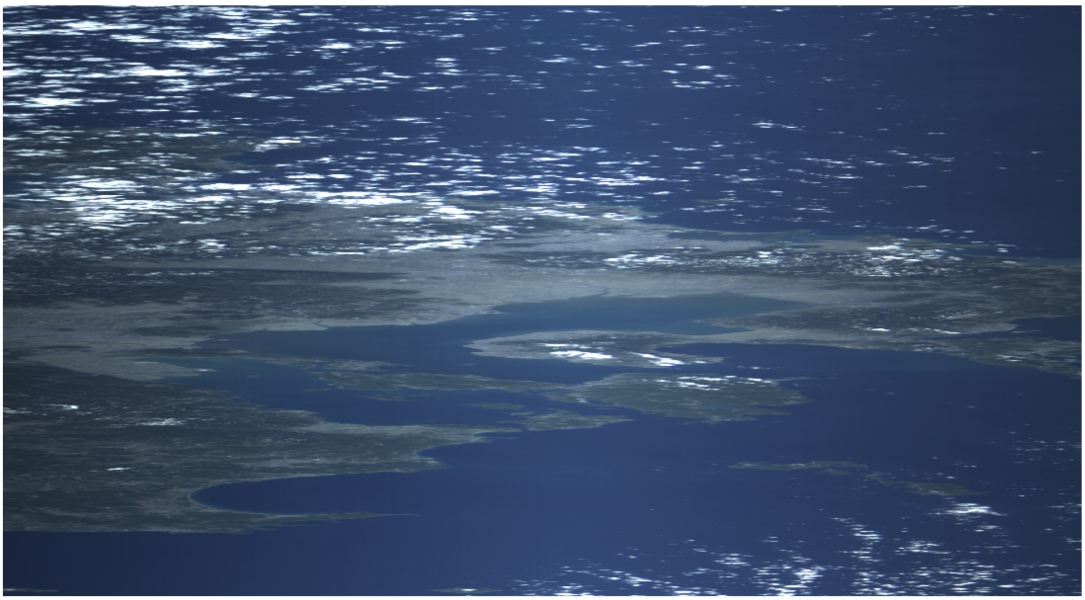
Original Image: aquawatchspencer_2025-01-03T01-17-50Z.png



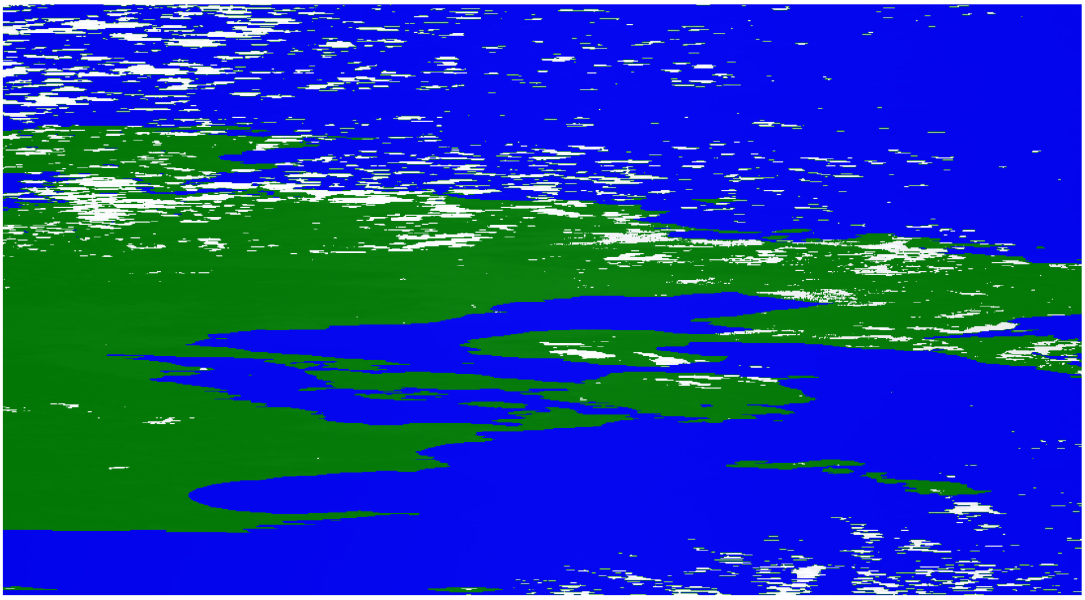
Labeled Image: aquawatchspencer_2025-01-03T01-17-50Z.png



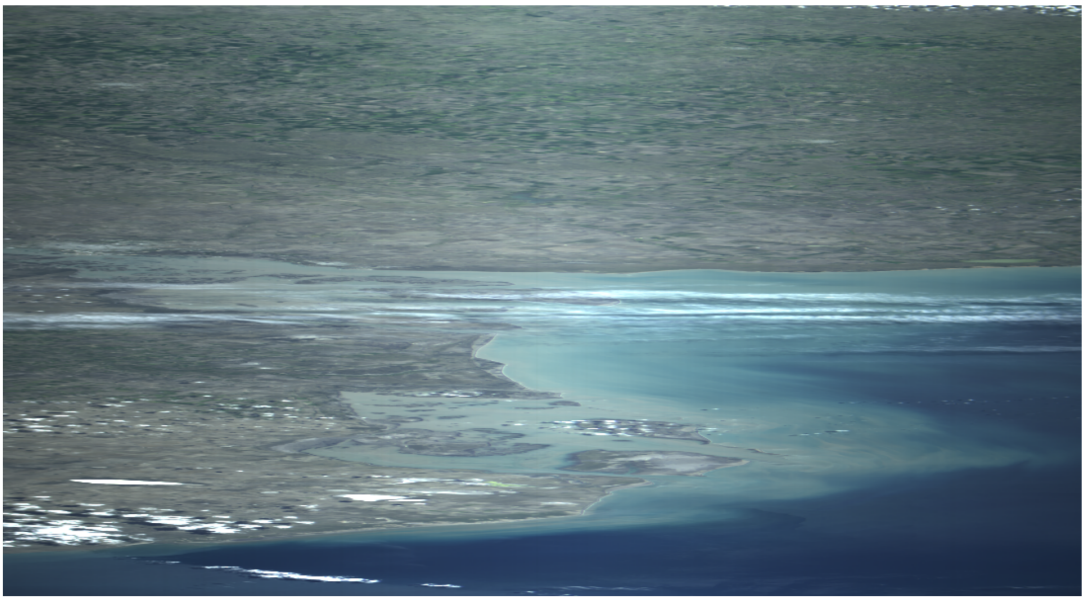
Original Image: ariake_2025-02-11T02-05-25Z.png



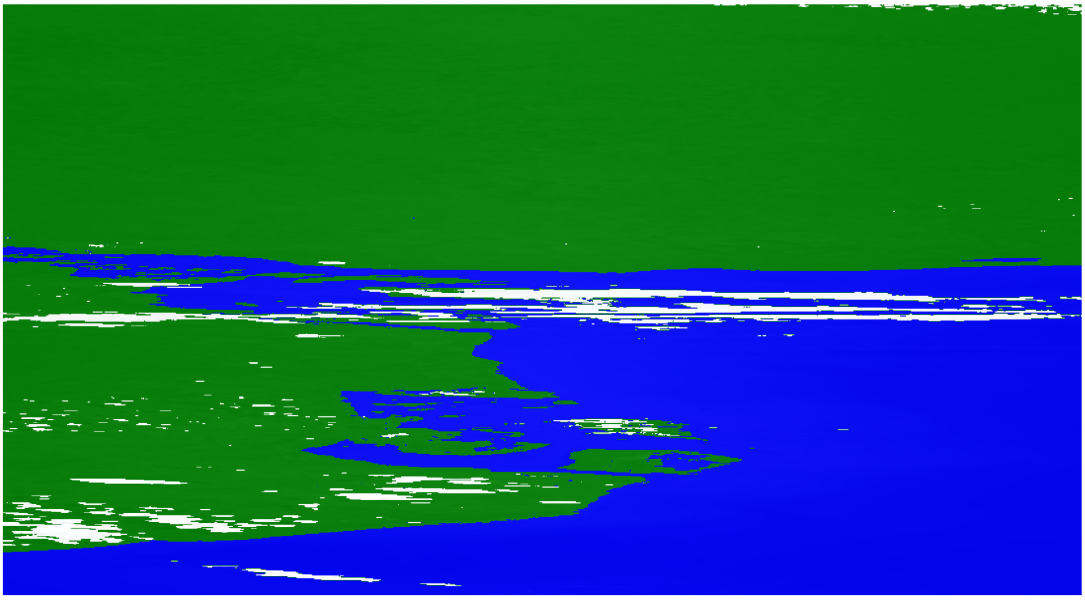
Labeled Image: ariake_2025-02-11T02-05-25Z.png



Original Image: blanca_2025-02-04T14-31-12Z.png

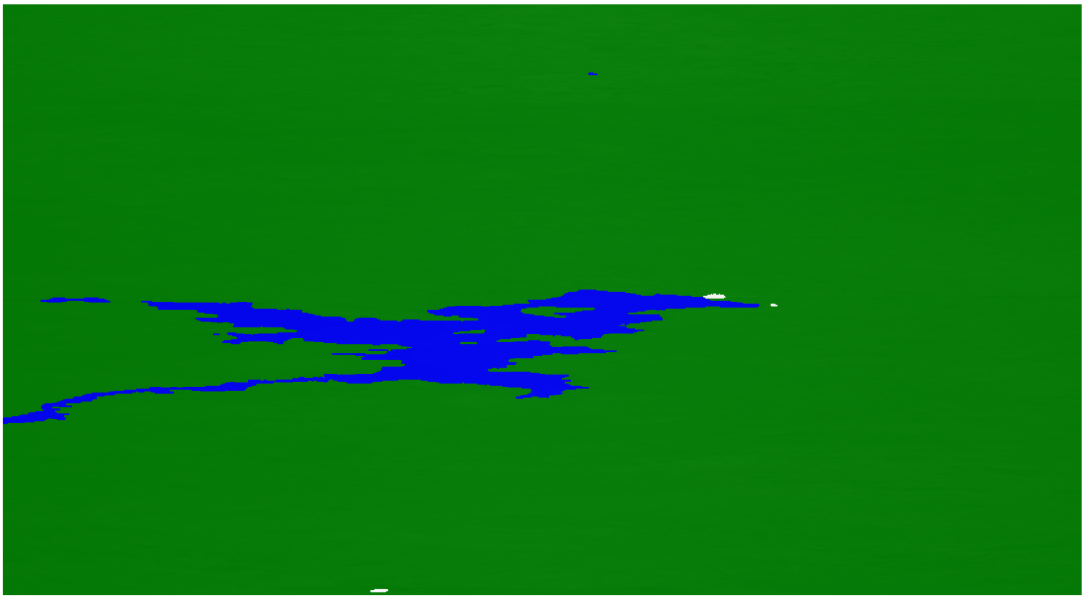


Labeled Image: blanca_2025-02-04T14-31-12Z.png

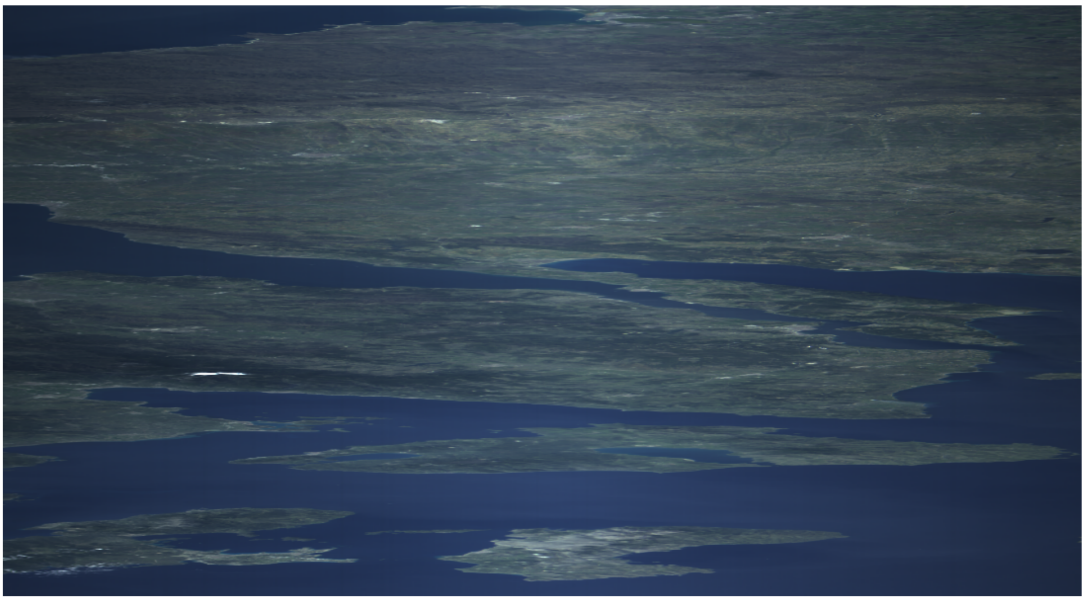


Original Image: bluenile_2025-01-25T08-23-16Z.png

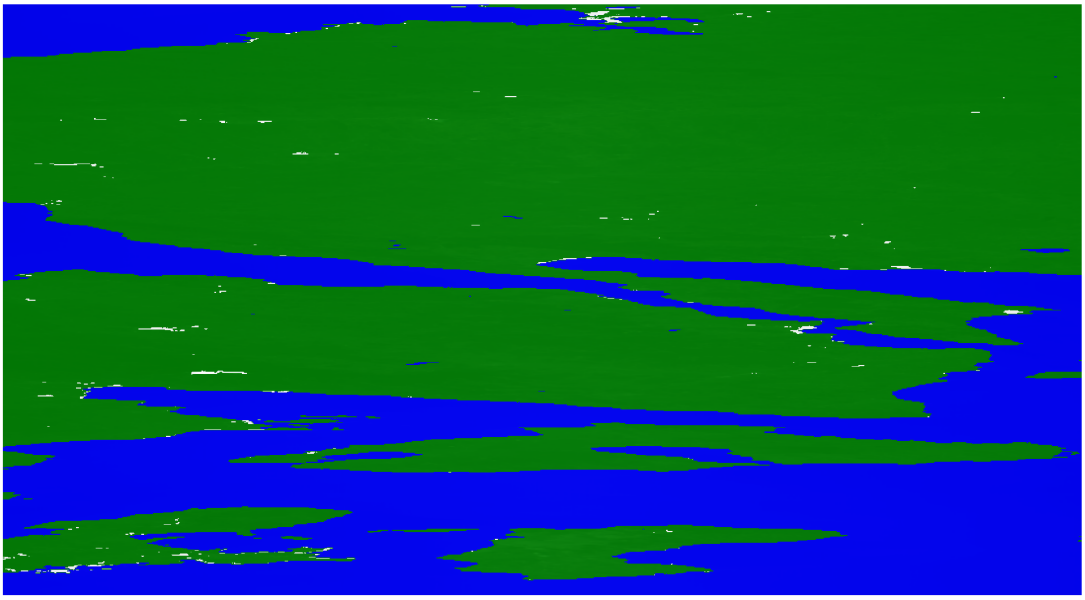
Labeled Image: bluenile_2025-01-25T08-23-16Z.png



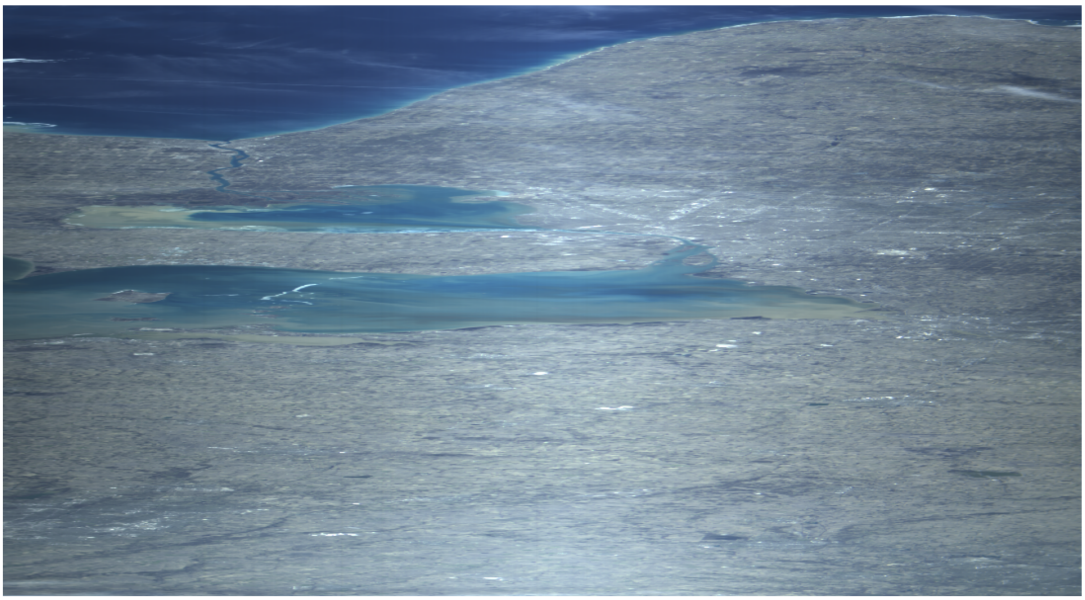
Original Image: dardanelles_2025-03-08T09-27-22Z.png



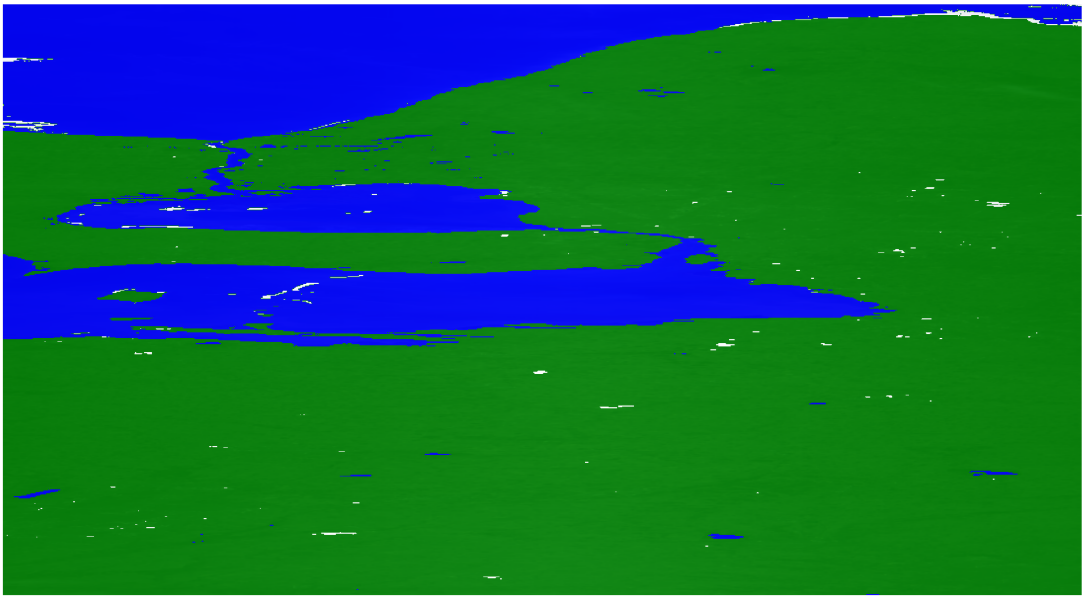
Labeled Image: dardanelles_2025-03-08T09-27-22Z.png



Original Image: erie_2025-03-13T16-22-46Z.png



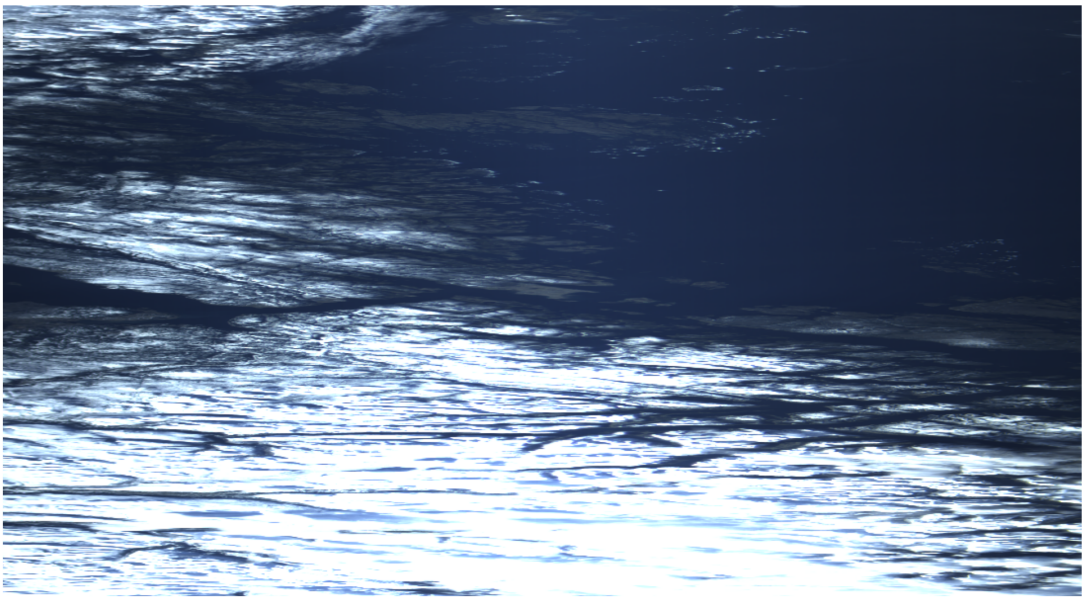
Labeled Image: erie_2025-03-13T16-22-46Z.png



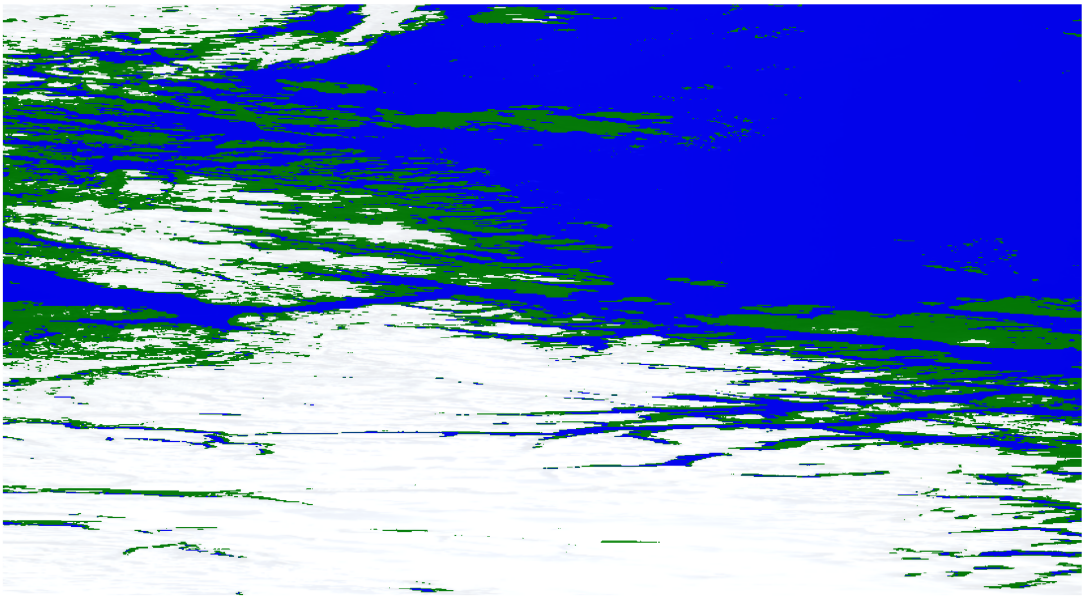
Original Image: falklandsatlantic_2025-03-03T14-11-51Z.png

Labeled Image: falklandsatlantic_2025-03-03T14-11-51Z.png

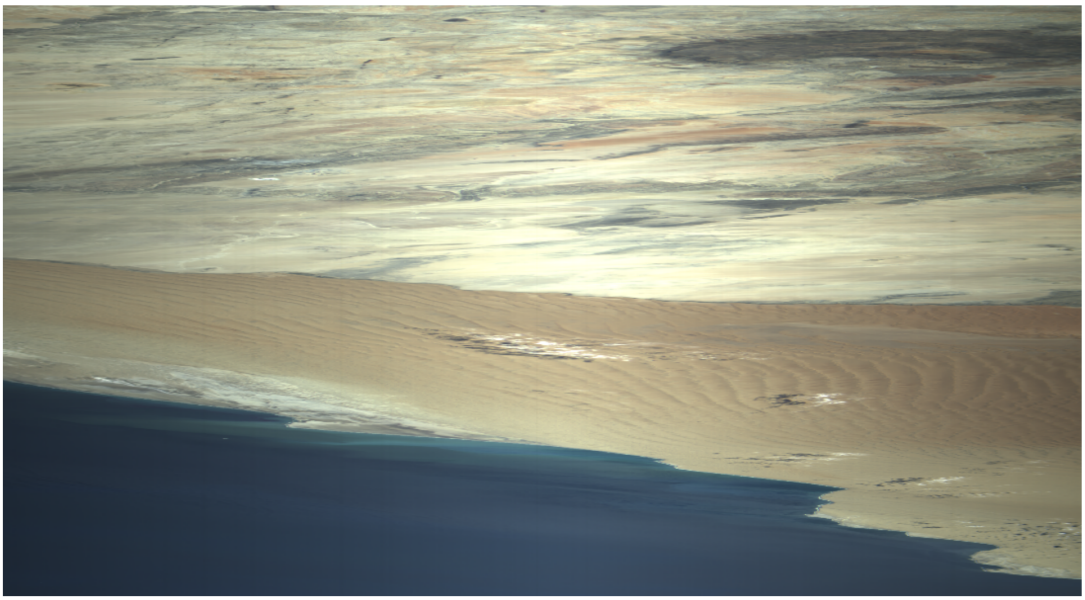
Original Image: frohavet_2025-02-25T11-26-39Z.png



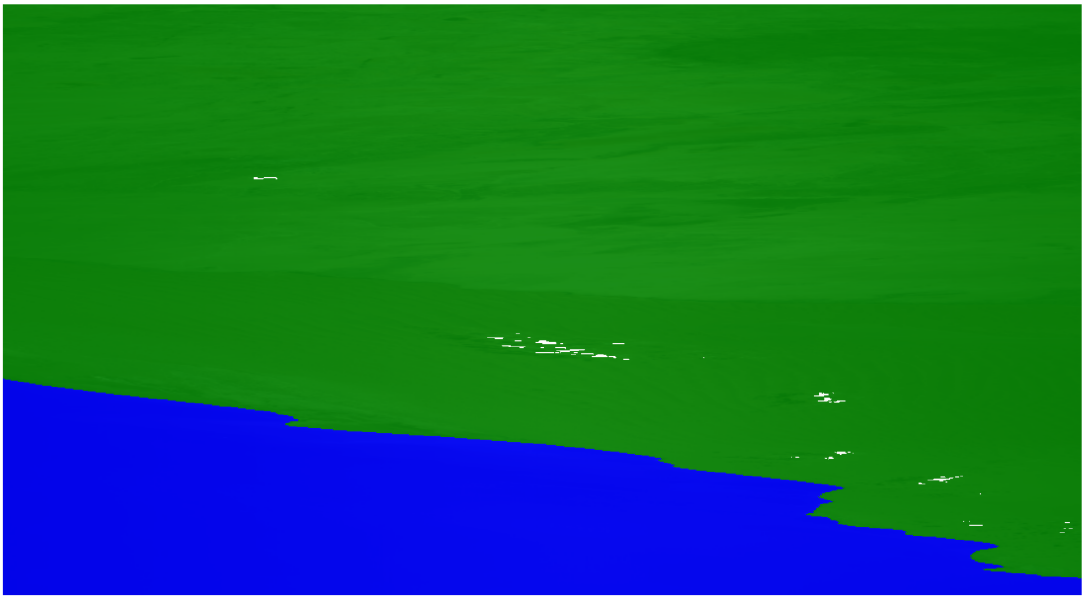
Labeled Image: frohavet_2025-02-25T11-26-39Z.png



Original Image: gobabeb_2025-02-02T09-24-52Z.png

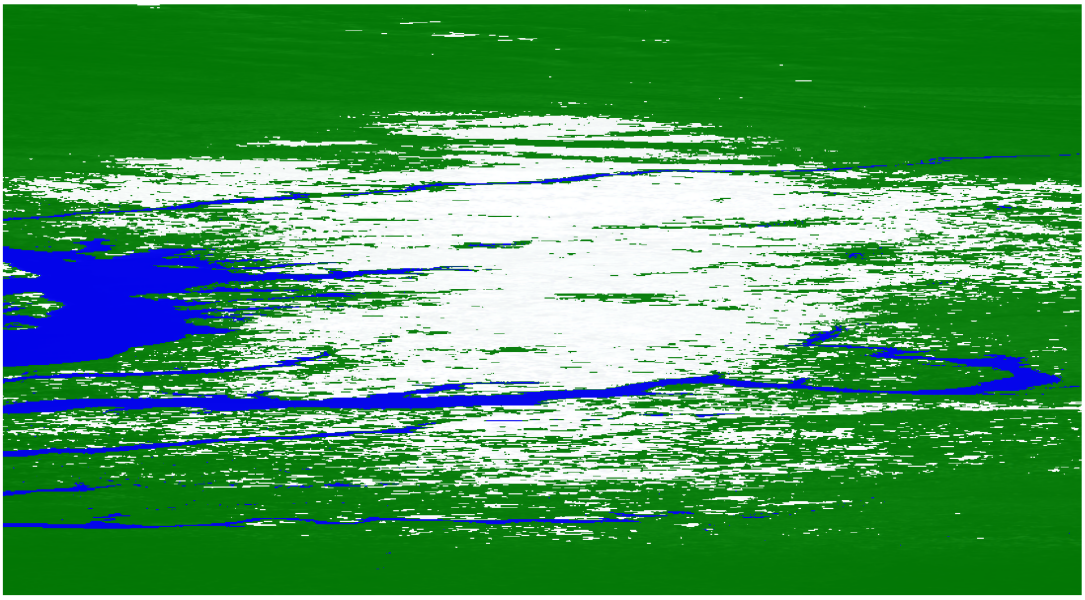


Labeled Image: gobabeb_2025-02-02T09-24-52Z.png

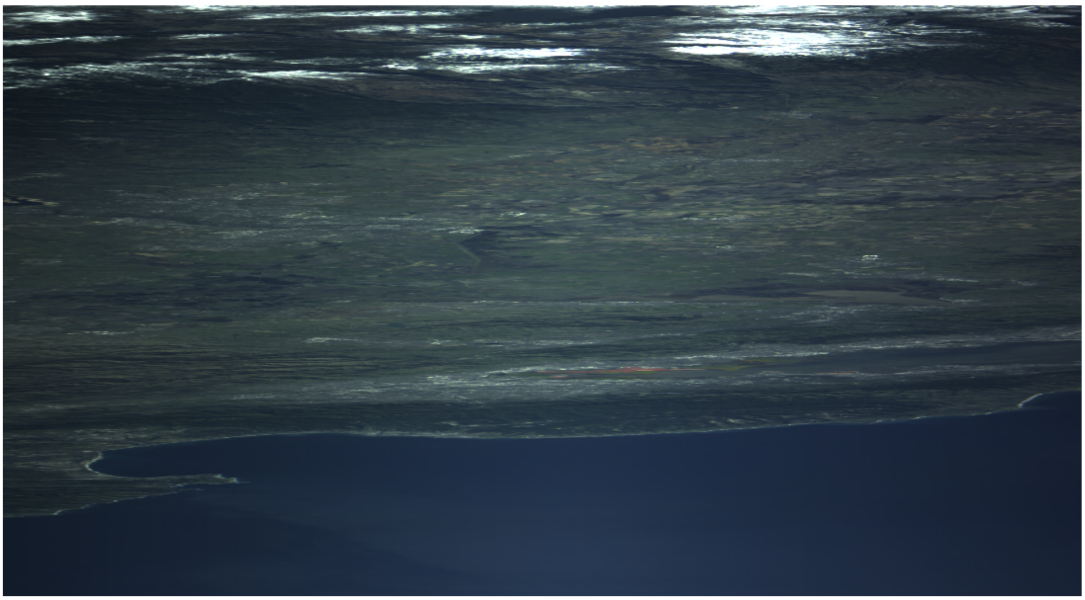


Original Image: goddard_2025-01-09T16-07-16Z.png

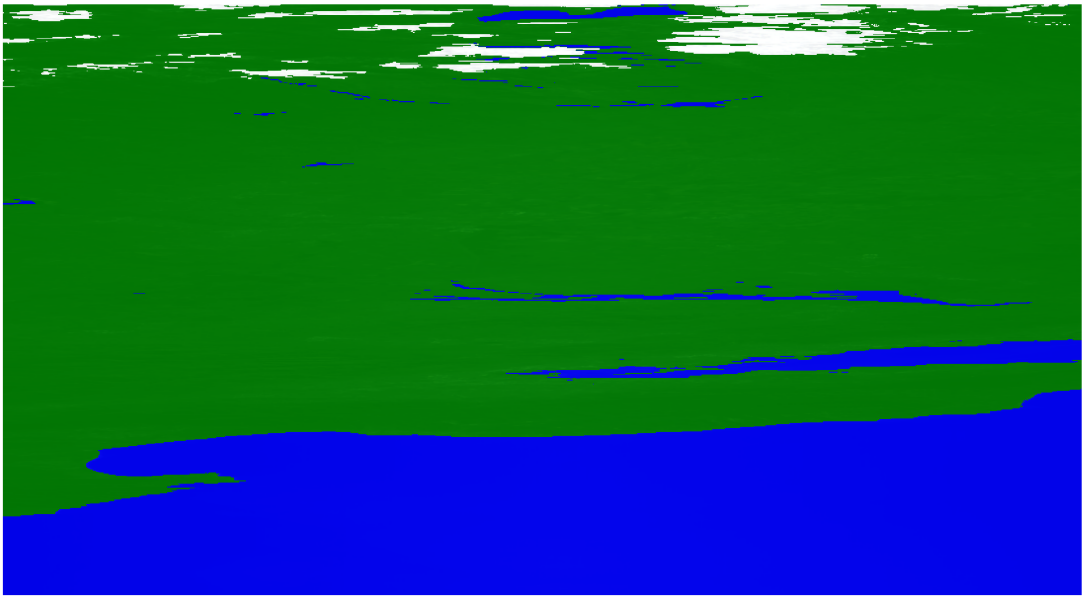
Labeled Image: goddard_2025-01-09T16-07-16Z.png



Original Image: grizzlybay_2025-01-22T19-11-18Z.png



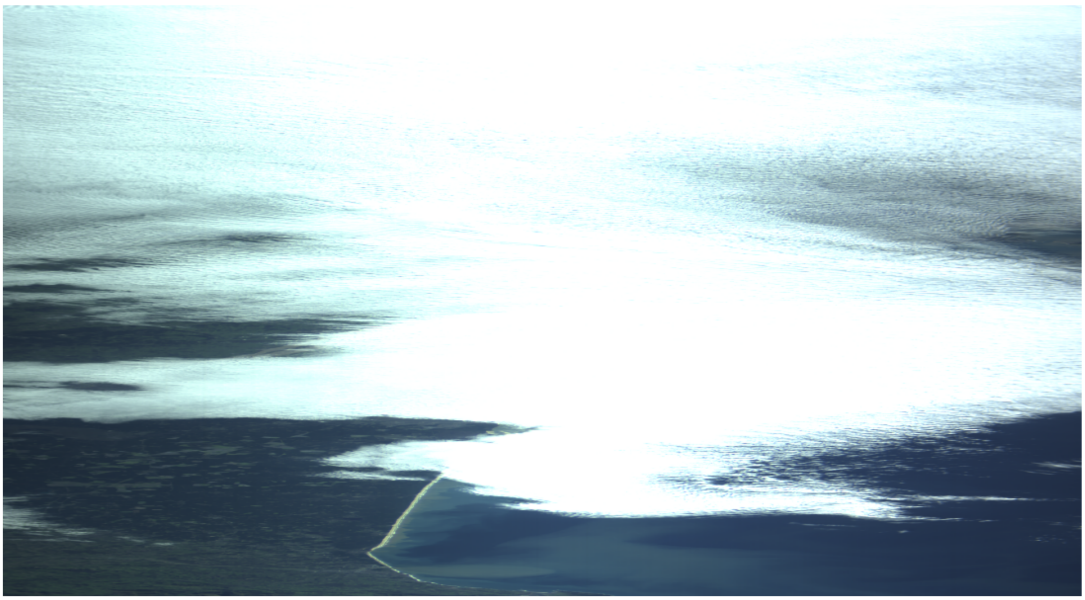
Labeled Image: grizzlybay_2025-01-22T19-11-18Z.png



Original Image: gulfofcalifornia_2025-01-14T18-19-49Z.png

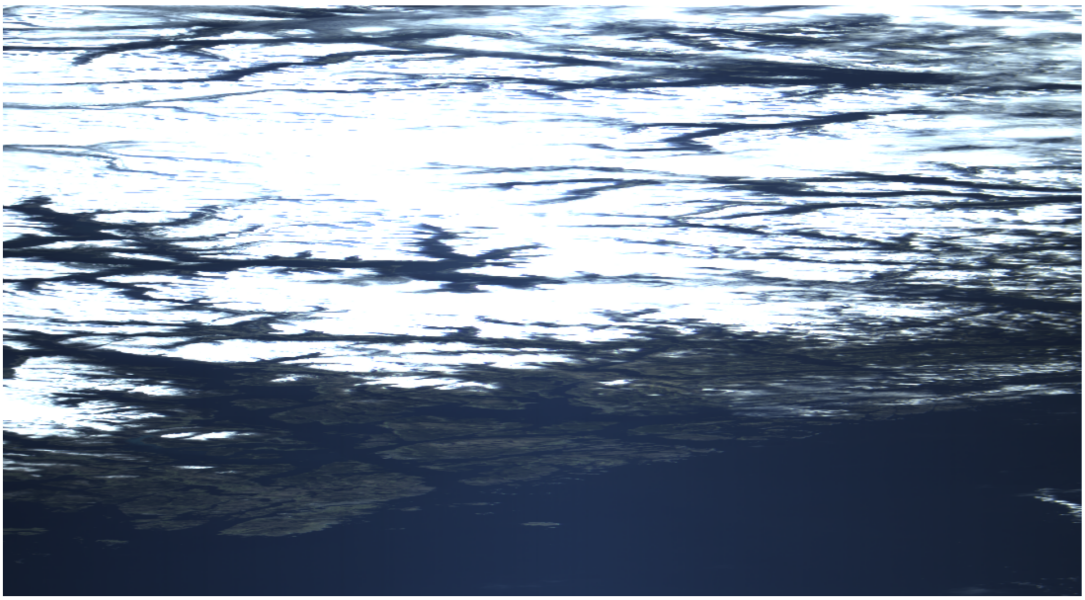
Labeled Image: gulfofcalifornia_2025-01-14T18-19-49Z.png

Original Image: hypernetEstuary_2024-12-28T11-29-35Z.png

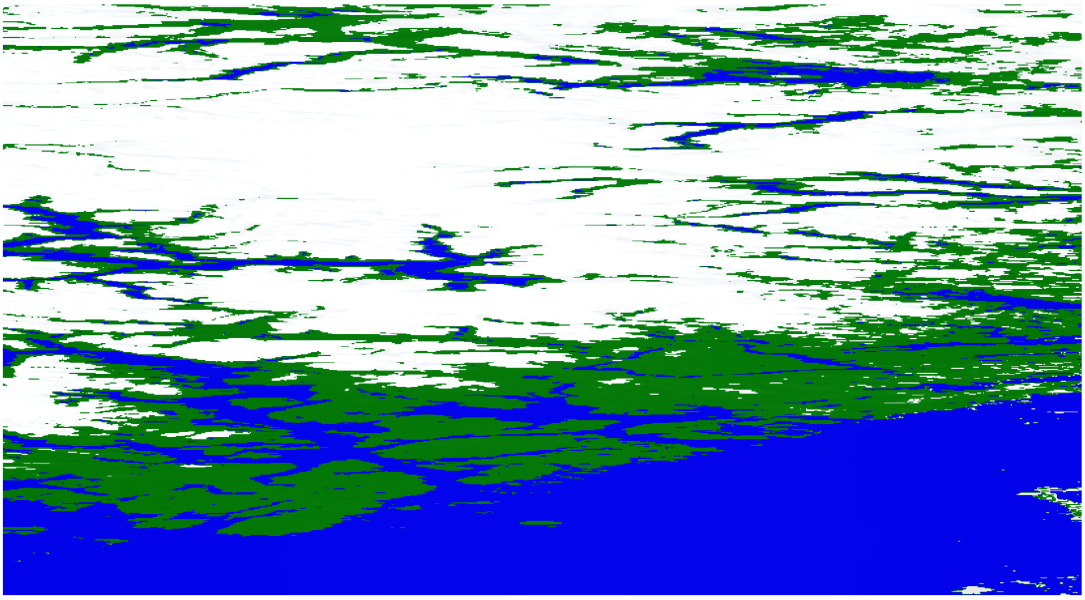


Labeled Image: hypernetEstuary_2024-12-28T11-29-35Z.png

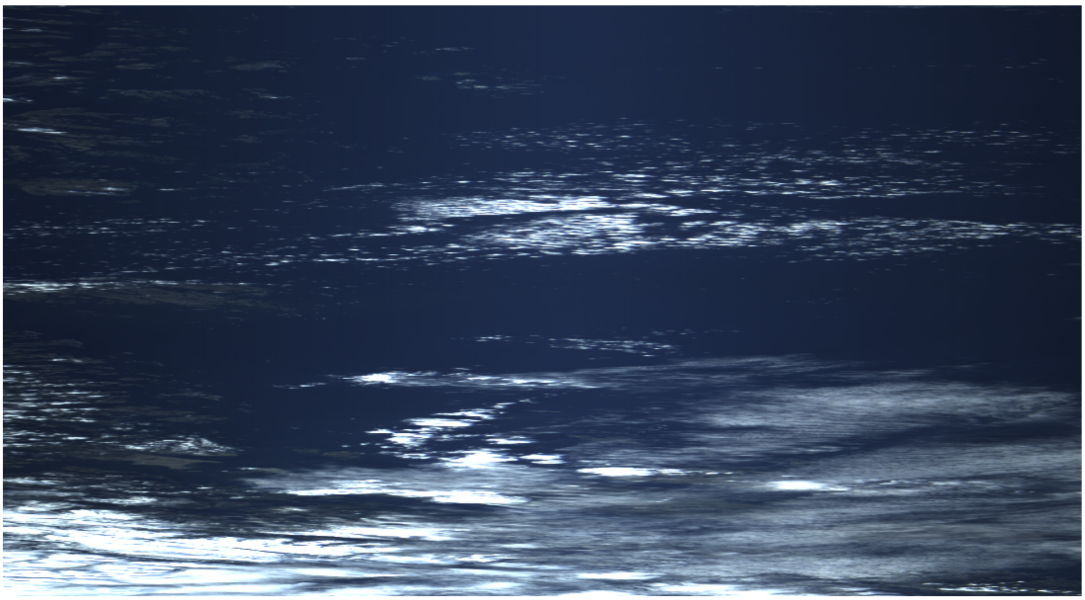
Original Image: image61N6E_2025-03-13T11-27-56Z.png



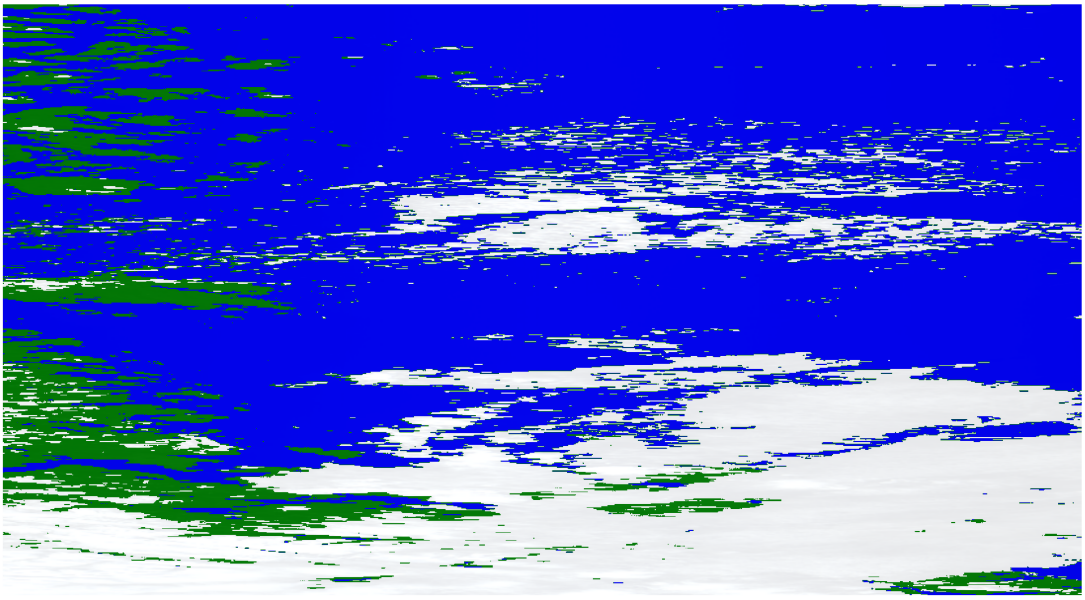
Labeled Image: image61N6E_2025-03-13T11-27-56Z.png



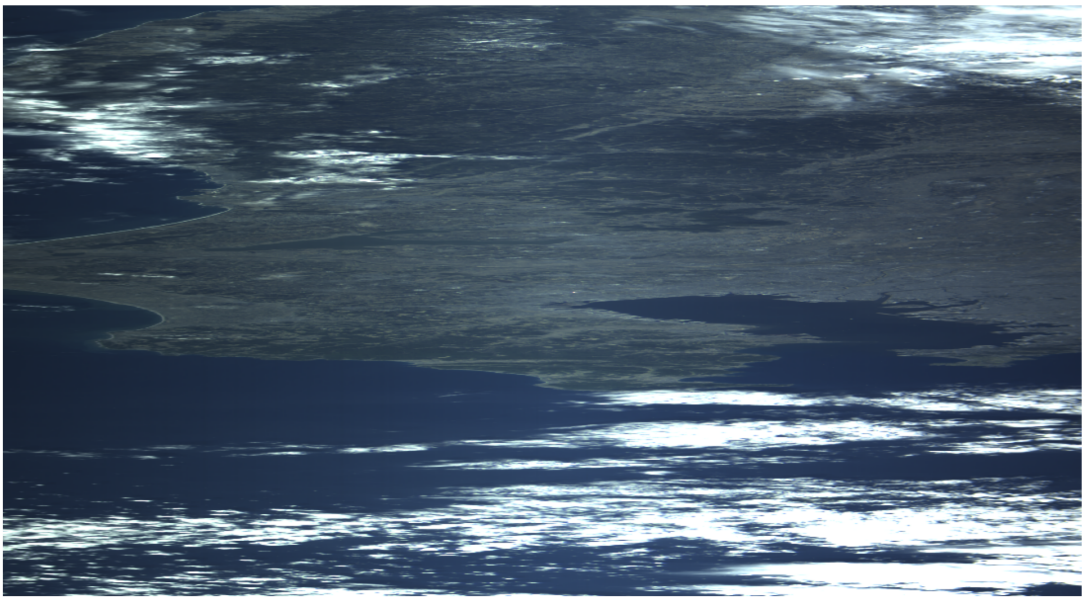
Original Image: image65N10E_2025-03-12T11-20-52Z.png



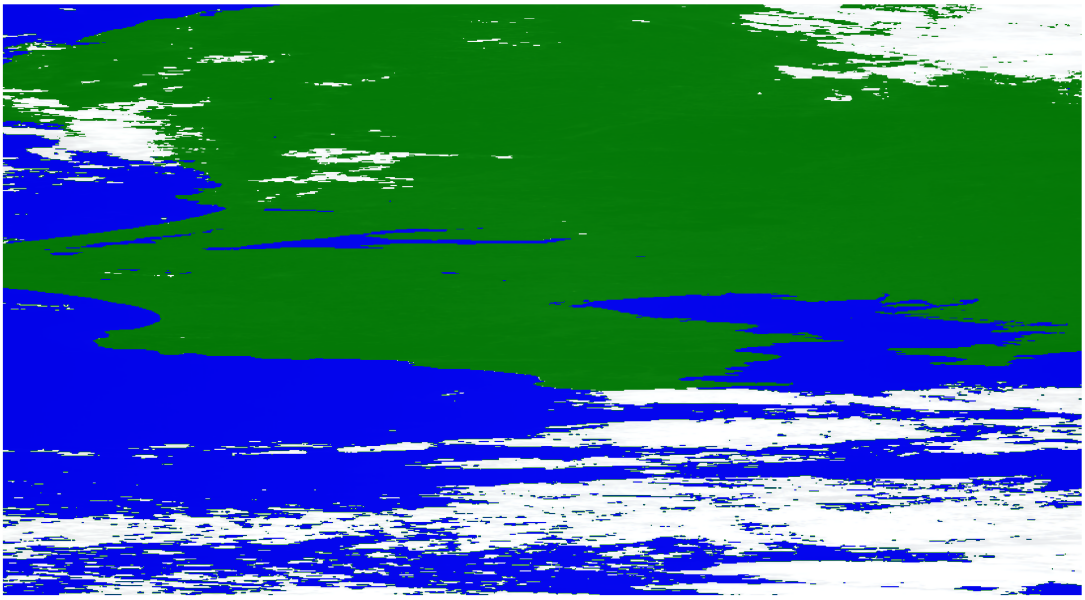
Labeled Image: image65N10E_2025-03-12T11-20-52Z.png



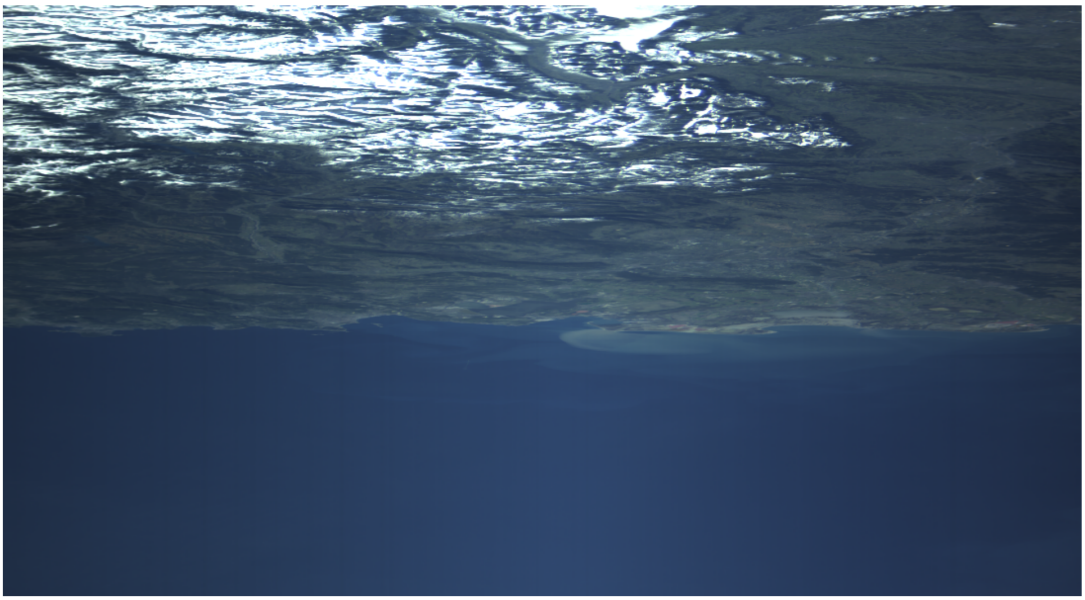
Original Image: kemigawa_2025-01-22T01-31-13Z.png



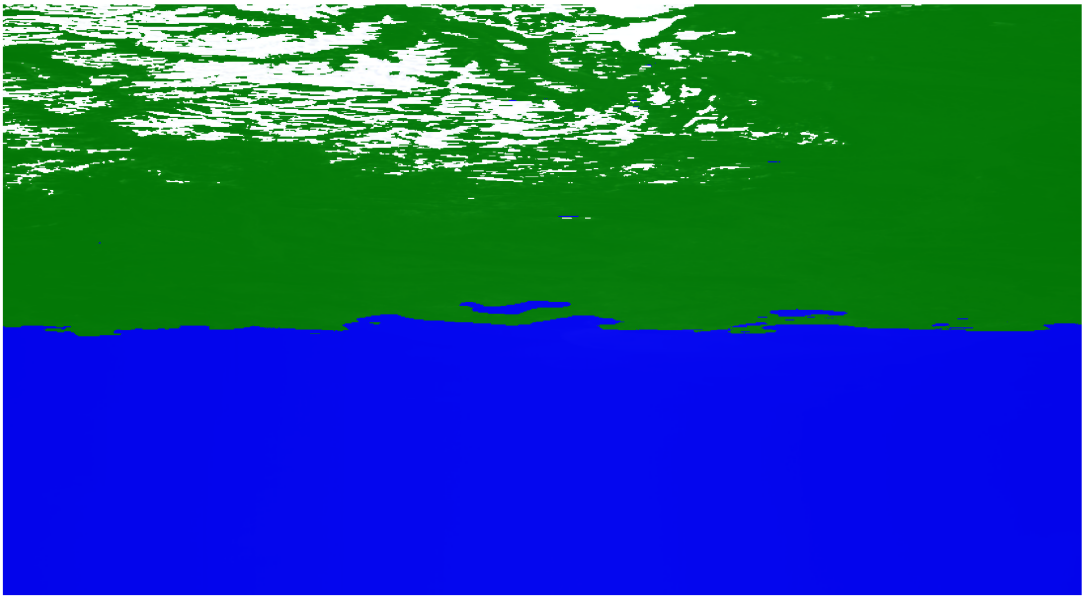
Labeled Image: kemigawa_2025-01-22T01-31-13Z.png



Original Image: lacrau_2024-12-26T11-15-54Z.png

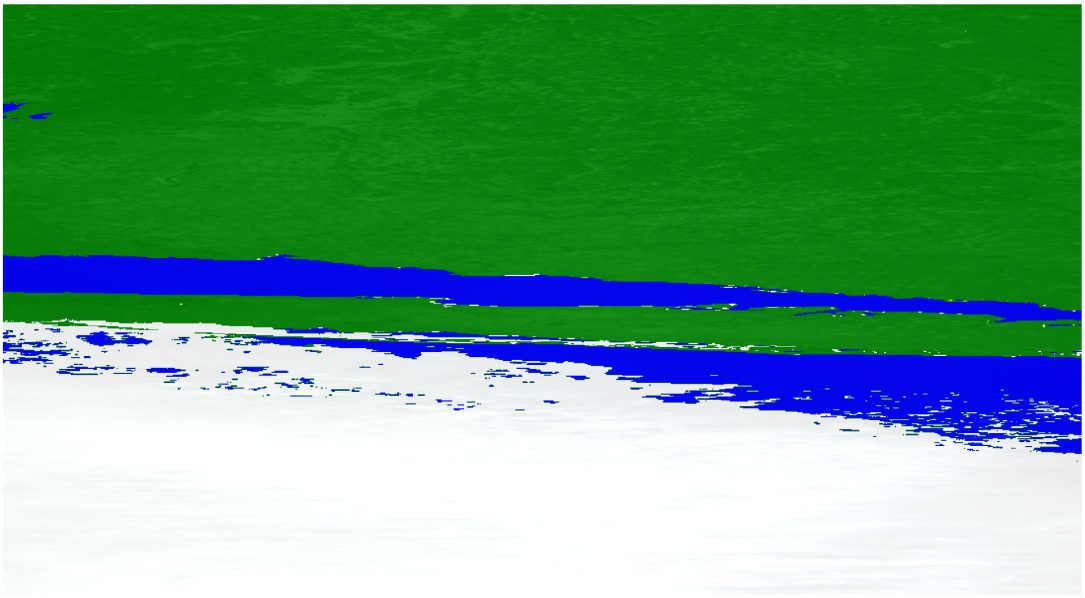


Labeled Image: lacrau_2024-12-26T11-15-54Z.png

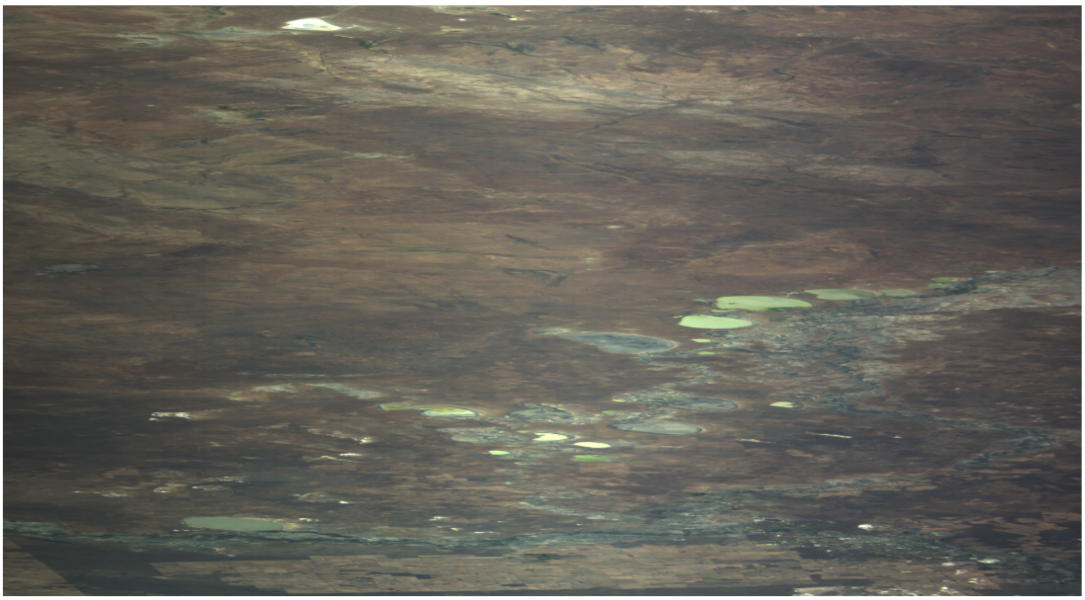


Original Image: longisland_2025-01-22T15-57-39Z.png

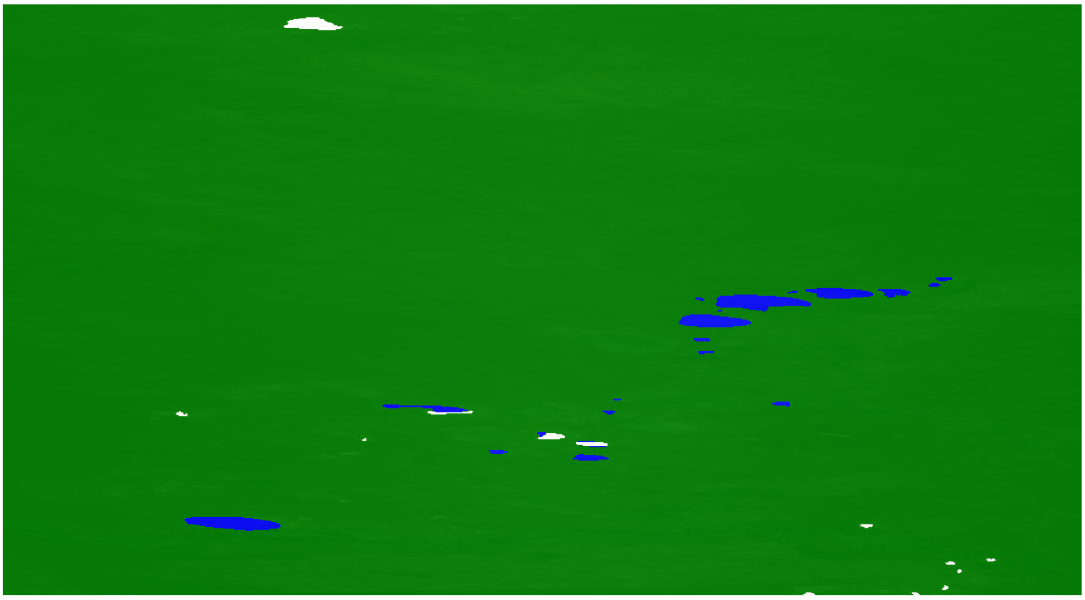
Labeled Image: longisland_2025-01-22T15-57-39Z.png



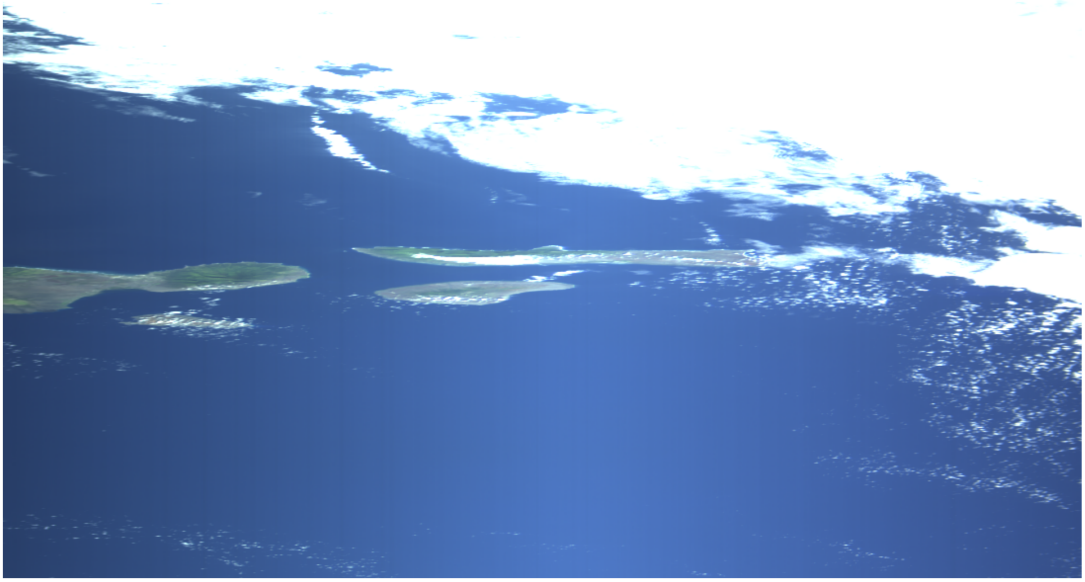
Original Image: menindee_2025-01-02T01-10-11Z.png



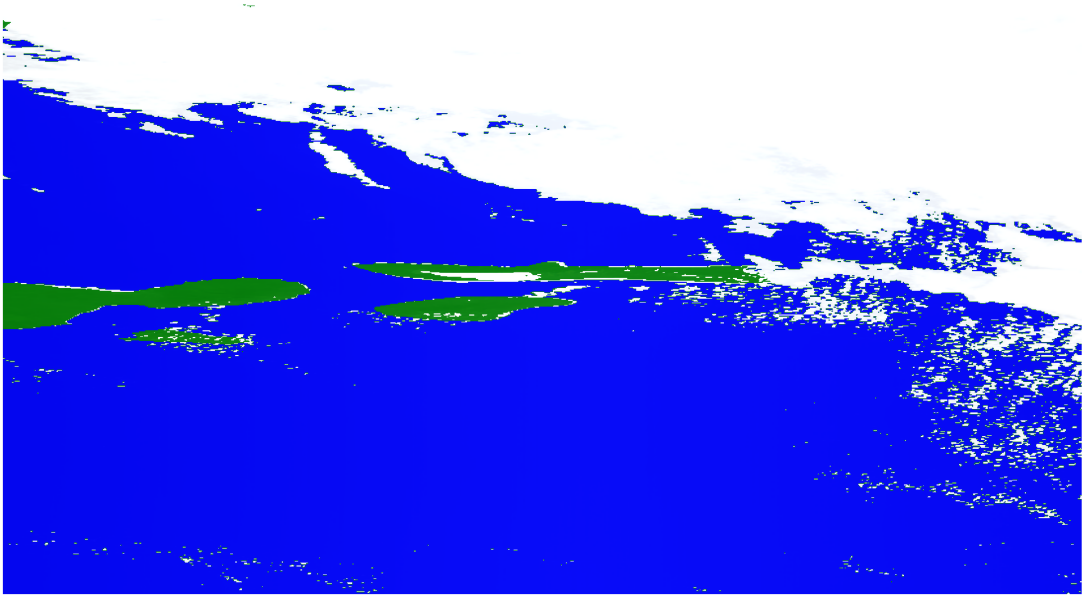
Labeled Image: menindee_2025-01-02T01-10-11Z.png



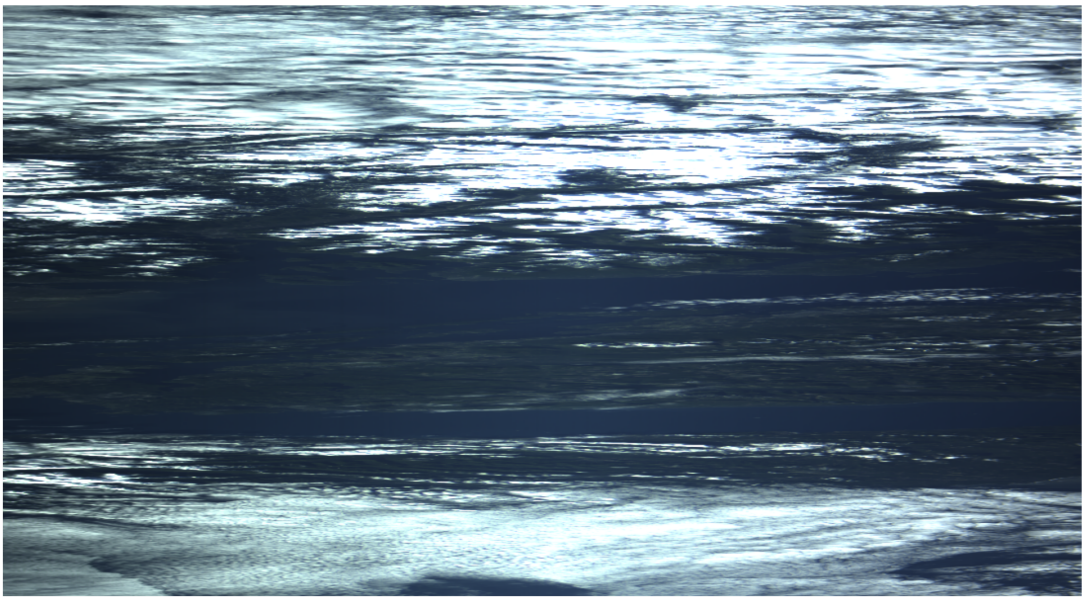
Original Image: moby_2025-01-08T20-54-59Z.png



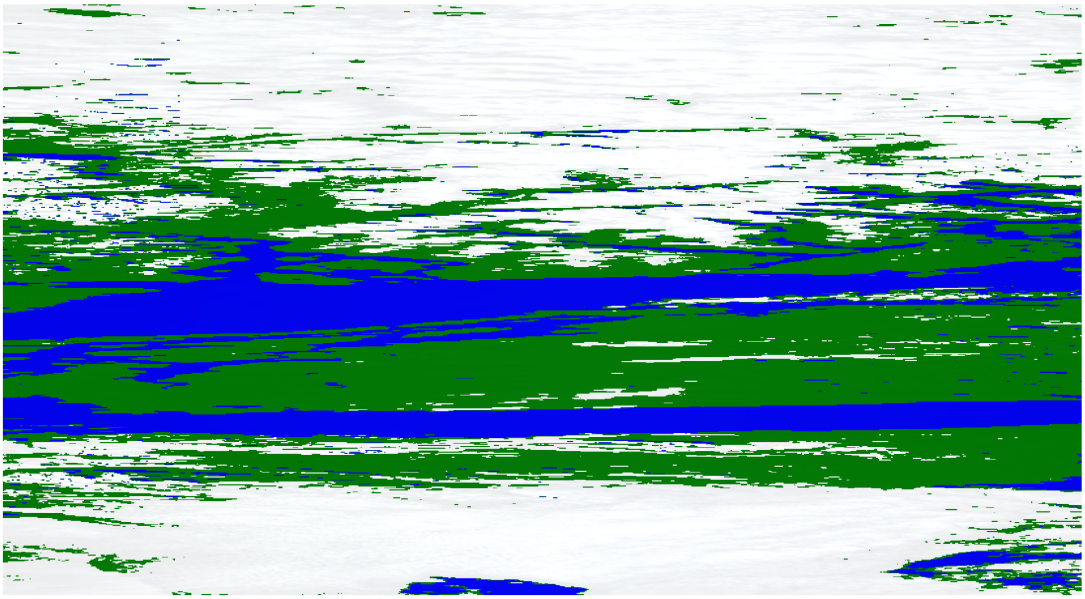
Labeled Image: moby_2025-01-08T20-54-59Z.png



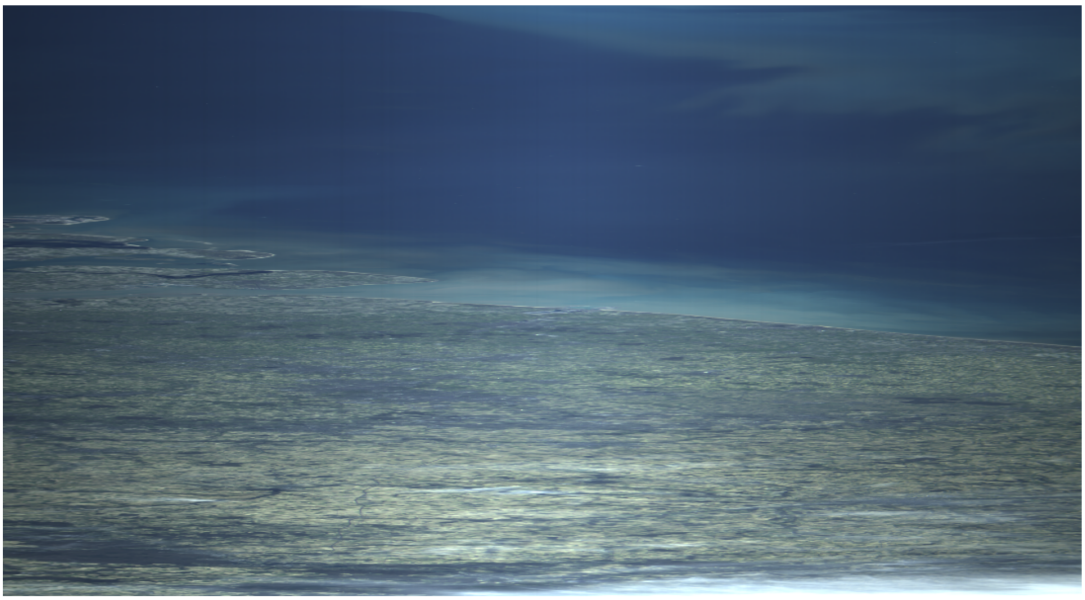
Original Image: straitofgeorgia_2025-01-24T19-21-34Z.png



Labeled Image: straitofgeorgia_2025-01-24T19-21-34Z.png



Original Image: zeebrugge_2025-03-08T11-00-54Z.png



Labeled Image: zeebrugge_2025-03-08T11-00-54Z.png